MEßWERTBASIERTE QUALITÄTSSICHERUNG

– Ein generisches Distanzmaß zur Erweiterung bisheriger Softwareproduktmaße –

Von der Fakultät für Mathematik,
Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
(Dr. rer.nat.)

genehmigte Dissertation

vorgelegt von

Dipl.-Inform.
Frank Simon

geboren am 11.4.1969 in Osnabrück

Erster Gutachter:	Prof. Dr. C. Lewerentz
Zweiter Gutachter:	Prof. Dr. R. Dumke
Dritter Gutachter:	Prof. Dr. H. Schmidt

Tag der mündlichen Prüfung: 12. Juli 2001

Inhaltsverzeichnis

1 EINLEITUNG/ÜBERSICHT 1

1.1 PROBLEMBEREICH 1
1.2 LÖSUNGSANSÄTZE IN DIESER ARBEIT 2
1.3 ORGANISATORISCHE KONTEXT DIESER ARBEIT 4

2 SOFTWAREQUALITÄT 5

2.1 FIXE QUALITÄTSMODELLE 7
2.2 EINFLUßFAKTOREN FÜR EIN ANGEPAßTES QUALITÄTSMODELL 11
2.3 PROZESSE ZUR DEFINITION EINES ANGEPAßTEN QUALITÄTSMODELLS 13
2.3.1 Goal-Question-Metric-Methode 14
2.3.2 Modifikationen der Goal-Question-Metric-Methode: Erfahrungen 15
2.4 ZUSAMMENFASSUNG 17

3 SOFTWAREMAßE 19

3.1 KLASSIFIZIERUNG 21
3.1.1 Prozeßmaße 21
3.1.2 Ressourcenmaße 23
3.2 PRODUKTMAßE 23
3.2.1 Größenmodellmaße 25
3.2.2 Kopplungsmodelle 26
3.2.3 Kohäsionsmodelle 27
3.3 PROZESSE FÜR EINEN ERFOLGREICHEN PRODUKTMAßEINSATZ 28
3.3.1 Beurteilung von Produktqualität 29
3.3.2 Vorbereitung eines Reviews 31
3.3.3 Überarbeitung eines Programms 33
3.3.4 Trendanalyse 35
3.4 MEßWERKZEUGE 36
3.4.1 Beispiel eines Softwareprozeß-Meßwerkzeuges 38
3.4.2 Beispiel eines Softwareressourcen-Meßwerkzeuges 39
3.4.3 Beispiele von Softwareprodukt-Meßwerkzeugen 40
3.4.4 Datrix 42
3.5 CROCODILE 44
3.6 PROBLEME BEIM PRODUKTMAßEINSATZ 47
3.6.1 Allgemeine Probleme beim Produktmaßeinsatz 48
Vorgehen 48
Management 49
Promotoren 50
Die Betroffenen 50
Ausbildung und Unterstützung 51
Pilotprojekte 52
Werkzeuge 53
3.6.2 Spezifische Probleme beim Produktmaßeinsatz 54
3.6.3 Weitere Zielsetzung dieser Arbeit 56
3.7 ZUSAMMENFASSUNG 58

4 MEßTHEORIE 59

4.1 MERKMALE UND RELATIONEN 61
4.1.1 Produktmodelle 62
4.1.2 Relationen auf Produktmodellen 66
Relationen auf Produktmodellen mit qualitativen Merkmalen 66
Relationen auf Produktmodellen mit quantitativen Merkmalen 68
4.1.3 Relationensysteme 69
4.2 SKALEN 70
4.2.1 Nominalskala 72
4.2.2 Ordinalskala 72
4.2.3 Intervallskala 73
4.2.4 Rationalskala 73
4.2.5 Absolutskala 74
4.3 VALIDIERUNG 74
4.3.1 Interne Validierung 76
4.3.2 Externe Validierung 79
4.4 PROZEß ZUR MEßTHEORETISCH KONFORMEN SKALENDEFINITION 83
4.5 ZUSAMMENFASSUNG 86

5 GRAPHENMODELL BISHERIGER MAßE 87

5.1 MESSEN ALS KNOTENATTRIBUTIERUNG 88
5.1.1 Vermessen von attributierten Graphen mit Größenmaßen 89
5.1.2 Attributierte Graphen von Kopplungsabstraktionen 90
5.1.3 Attributierte Graphen von Kohäsionsabstraktionen 91
5.2 ANWENDUNG DES GRAPHENMODELLS 91
5.3 KOMBINATION VON STRUKTUR UND MEHREREN MEßDATEN 94
5.3.1 Monosichten 95
5.3.2 Multisichten 97
5.4 KANTENATTRIBUTIERUNGEN INNERHALB DES GRAPHENMODELLS 101
5.4.1 Bekannte Kantenattributierungen 102
5.5 ZUSAMMENFASSUNG 104

6 DISTANZ- UND ÄHNLICHKEITSKONZEPT 105

6.1 ÄHNLICHKEIT ZWISCHEN ENTITÄTEN 106
6.2 DISTANZEN ZWISCHEN ENTITÄTEN 109
6.3 MERKMALBASIERTE SKALEN 112
6.3.1 Ordinalskalierte Kantenvermessung 113
6.3.2 Intervallskalierte Knotenvermessung 115
6.4 TECHNIKEN FÜR VERWENDUNG VON DISTANZMEßWERTEN 116
6.4.1 Clusteranalyse 117
6.4.2 Visualisierung 120
Nichtlineares Optimierungsverfahren 121
Hauptkomponentenanalyse 124
6.5 ZUSAMMENFASSUNG 128

7 DISTANZMAßBASIERTE KOHÄSION 129

7.1 KOHÄSION 130
7.1.1 Funktionale Kohäsion 132
7.1.2 Abstrakte Kohäsion 138
7.2 KOPPLUNG 143
7.2.1 Interaktionskopplung 144
7.2.2 Komponentenkopplung 145
7.2.3 Vererbungskopplung 146
7.3 PROZEß ZUR DISTANZMAßBASIERTEN KOHÄSIONSBESTIMMUNG 147
7.3.1 Prozeß zur kopplungbasierten Kohäsionsbestimmung mittels des generischen Distanzmaßes 148
7.3.2 Allgemeiner Prozeß zur distanzmaßbasierten Kohäsionsbestimmung 152
7.4 ZUSAMMENFASSUNG 157

8 KOHÄSION OBJEKTORIENTIERTER SYSTEME 159

8.1 ABSTRAKTIONSNIVEAUS OBJEKTORIENTIERTER SYSTEME 160
8.2 BERÜCKSICHTIGUNG VON VERERBUNG 163
8.2.1 Generalisierung/Spezialisierung 164
8.2.2 Schnittstellenabstraktion 165
8.2.3 Typhierarchieimplementierung 166
8.2.4 Quelltextwiederverwendung 167
8.3 KLASSENKOPPLUNGBASIERTE KOHÄSION 167
8.3.1 Vererbungbasierte Kohäsion 167
Ungewichtete Vererbung 168
Gewichtete Vererbung 172
8.3.2 Interaktionbasierte Kohäsion 175
Ungewichtete Interaktion 176
Gewichtete Interaktion 179
Gewichtete Interaktion: Externe Kopplung 179
Gewichtete Interaktion: Datenbasierte/strukturelle Kopplung 181
8.4 ATTRIBUTS- UND METHODENKOPPLUNGBASIERTE KOHÄSION 183
8.5 ALLGEMEINE MERKMALBASIERTE KOHÄSION 186
8.6 ZUSAMMENFASSUNG 190

9 EXTERNE VALIDIERUNG DES GENERISCHEN KOHÄSIONSKONZEPTS 191

9.1 CROCOCOSMOS 192
9.2 EXTERNE VALIDIERUNG INNERHALB DES PROZESSES „BEURTEILUNG VON PRODUKTQUALITÄT“ 198
9.2.1 Prozeßdurchlauf für die JWAM-Version 1.4 202
Beurteilungsvorbereitung 203
Beurteilungsdurchführung 206
Beurteilungsreflexion 207
9.2.2 Prozeßdurchlauf für die JWAM-Version 1.5 209
Beurteilungsvorbereitung 209
Beurteilungsdurchführung 211
Beurteilungsreflexion 211
9.2.3 Erster Prozeßdurchlauf für das Versicherungssystem VS 214

Beurteilungsvorbereitung ... 215
Beurteilungsdurchführung ... 217
Beurteilungsreflexion ... 218
9.2.4 Zweiter Prozeßdurchlauf für das Versicherungssystem VS ... 220
Beurteilungsvorbereitung ... 220
Beurteilungsdurchführung ... 223
Beurteilungsreflexion ... 224
9.3 EXTERNE VALIDIERUNG INNERHALB DES PROZESSES ÜBERARBEITUNG EINES PROGRAMMS ... 227
9.3.1 Semiautomatische Gruppierung von Klassen ... 227
Semiautomatische Gruppierung von Crocodile ... 228
9.3.2 Identifikation Refactoring-werter Softwareteile ... 233
Verschieben einer Methode ... 235
Verschieben eines Attributs ... 236
Separieren einer Klasse / Zusammenführen zweier Klassen ... 237
Entfernung/Einführen von Delegationen ... 239
Erfahrungen beim Identifizieren von Refactoring-bedürftigen Softwareteilen ... 242
9.4 ZUSAMMENFASSUNG ... 246

10 ZUSAMMENFASSUNG/AUSBLICK ... 247

11 REFERENZEN ... 257

12 ANHANG ... 271

Zusammenfassung

In dieser Arbeit werden neue Ansätze zur meßwertbasierten Qualitätssicherung von Softwareprodukten für praxisrelevante Systemgrößen beschrieben. Dies umfaßt Hilfen zur spezifischen Definition von Qualität, Prozesse zur Qualitätssicherung, Werkzeugunterstützung sowie die Identifikation von Problemen bei der Anwendung vieler bisheriger Softwaremaße. Die pragmatische Anwendung der Meßtheorie erlaubt eine Erweiterung in Form eines auf Ähnlichkeiten zwischen Softwareelementen basierenden generischen Distanzmaßes. Dessen unterschiedliche Instanziierungen ermöglichen die automatische, anpaßbare Gruppierung von Elementen, sowie die ausdrucksstarke, ausschließlich durch Meßwerte bestimmte Softwarevisualisierung. Die ebenfalls in dieser Arbeit wiedergegebene empirische Arbeit, in der die Effizienz und Effektivität dieses Ansatzes mehrfach belegt wird, demonstriert die mögliche Bedeutung der meßwertbasierten Qualitätssicherung bei der Analyse großer Systeme.

Abstract

In this work new concepts for the metrics based quality assurance of large software systems are presented. This includes techniques for defining quality, processes for quality assurance, tool support and identification of problems when applying state-of-the-art software metrics. The pragmatic application of measurement theory allows an extension that is based on a special similarity measure for software elements. The application of this metric in different ways allows for automatic and adjustable grouping of software elements and the powerful software visualisation that is based on metric values only. The empirical work, which is also presented, shows the efficiency and effectivity of the new approach and demonstrates the potential importance of the metrics based quality assurance for the analysis of large software systems.

Tabellarische Darstellung des Lebens- und Bildungsganges

Name:	Frank Simon
geboren:	11.4.1969
in:	Osnabrück
Eltern:	Dr. Wolfgang Simon, Annemarie Simon, geb. Ellmer
Geschwister:	Anke Simon (1970), Berno Simon (1980)
Ehefrau:	Dipl.-Inform. Bettina Simon, geb. Steen
Kinder:	Marie-Sophie Simon (1999)

Bildungsweg:

1975 - 1979	Grundschule Bissendorf, Kreis Osnabrück
1979 - 1981	Orientierungsstufe Dom, Osnabrück
1981 - 1988	Staatlich anerkanntes Gymnasium des Bistums Osnabrück, *Ursulaschule*
Mai 1988	Abitur
Juli 1988 - Sept. 1989	Grundwehrdienst, elektronische Kampfführung, beendet als Obergefreiter
Okt. 89 - Juli 96	Studium an der Carl von Ossietzky Universität Oldenburg, Studiengang: Diplom-Informatik
1990 - 1996	Dozent in der KVHS-Erwachsenenbildung (EDV), Wochenendintensivkurse, Seminare, Workshops
April 1990 - April 1995	Mitglied des Studentenparlaments und des Konzils der Universität Oldenburg
April 1991 - April 1992	Vorsitzender der Studentenvereinigung Ring Christlich Demokratischer Studenten (RCDS) Oldenburg
Mai 1991 - Mai 1992	Mitglied des höchsten, parlamentarischen Gremiums (Senat) der Universität Oldenburg
April 1994 - Juli 1996	Schatzmeister des RCDS-Oldenburg
Aug. 1994 - Sept. 1994	Werkstudent der Physikalisch-Technischen Bundesanstalt Braunschweig (PTB), Ergebnisse erschienen 1998 als Bericht PTB-6.42-98-1 „*SPHERE: A Program Package for Calculating the Neutron Response of Spherical Proton Recoil Proportional Counters*“
Feb. 1996 - Juli 1996	Diplomarbeit bei den Hamburger-Elektrizitätswerken (HEW): „*Summative Software- /Hardware-Evaluation anhand des Fallbeispiels ‚Betriebsführungssystem im Kernkraftwerk Brunsbüttel‘* “
Juli 1996	Diplom im Studienfach Informatik (Gesamtnote: sehr gut)
Seit Okt. 1996	Wissenschaftlicher Mitarbeiter an der Brandenburgischen Technischen Universität Cottbus (BTU) am Lehrstuhl Software-Systemtechnik von Prof. Dr. C. Lewerentz
Mai 2001	Abgabe der Dissertation „*Meßbasierte Qualitätssicherung: Ein generisches Distanzmaß zur Erweiterung bisheriger Softwareproduktmaße*“ und Eröffnung des Promotionsverfahrens.
12. Juli 2001	Abschlußprüfung zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer.nat) mit Summa cum laude bestanden. Formale Ausstellung der Doktorurkunde zur Zeit in Arbeit.

„Wer vom Ziel nicht weiß,
kann den Weg nicht haben"
(Morgenstern: „*Wir fanden einen Pfad*")

1 Einleitung/Übersicht

Die wachsende Komplexität heutiger Softwaresysteme erfordert sowohl für die konstruktiven Phasen des Entwicklungsprozesses von Software als auch für die verschiedenen Arten der Softwarewartung wie Fehlerbeseitigung, Anpassung und Perfektionierung spezielle Techniken. Mit der zunehmenden Verbreitung iterativer Softwareerstellungsprozesse, d.h. der möglichst frühzeitigen Erstellung von Teilfunktionalität mit anschließenden, wiederholten Überarbeitungen, die dem fertigen Produkt immer ähnlicher werden, und mit der zunehmenden Dauer des Einsatzes eines Softwareprodukts gewinnen speziell die Wartungsaktivitäten immer mehr an Bedeutung. Deren Zweck, nämlich das Erreichen und Halten einer hohen Produktqualität, zielt sowohl auf das nach außen beobachtbare Verhalten als auch auf den internen Aufbau des Produkts. Dem letzteren kommt dabei insbesondere aus wirtschaftlichen Gesichtspunkten eine große Bedeutung zu, da der Aufwand für Anpassungen der nach außen sichtbaren Funktionalität maßgeblich von der Qualität des internen Aufbaus abhängt. Der Fokus dieser Arbeit liegt in der auf dem Konzept der *Vermessung von Software* basierenden Sicherung und Verbesserung dieser internen Qualität.

1.1 Problembereich

Für diese Sicherung und Verbesserung der internen Qualität von Software ist eine genaue Beschäftigung mit dem Konzept der Softwarequalität erforderlich. In Kapitel 2 werden daher bekannte, die Softwarequalität behandelnde Ansätze in Form von sogenannten *Qualitätsmodellen* kurz vorgestellt. Dies schließt sowohl fertige Qualitätsmodelle, die Identifikation verschiedener Dimensionen eines Qualitätsmodells, jeweils auf diese Dimensionen einwirkende, kontextabhängige Einflußfaktoren sowie Prozesse zur Erstellung eines angepaßten Qualitätsmodells ein.
In der Praxis zeigen sich allerdings bereits während dieser Aktivitäten, die im Vorfeld jeder praktisch durchgeführten Qualitätssicherung getätigt werden müssen, große Probleme, da

- eine Vielzahl unterschiedlicher Qualitätsmodelle unterschiedlicher Qualitätsmodellarten (fixe, analytische oder prozeßorientierte Modelle zur Festlegung eines Qualitätsverständnisses) existiert, von denen jede Variante den Anspruch auf Korrektheit besitzt,
- für den Bereich der Softwarequalität häufig eine uneinheitliche und zum Teil widersprüchliche Nomenklatur verwendet wird, die ein Vergleichen der verschiedenen Modelle deutlich erschwert, und da
- alle Modelle aufgrund der Betonung auf die Einfachheit der Anwendbarkeit und das damit verbundene Ausblenden des hinter einer Anwendung stehenden Zwecks in der Praxis häufig zu restriktiv sind und dadurch innerhalb eines konkreten Anwendungskontextes eben nicht immer direkt anwendbar sind.

Um das Ziel der Verbesserung der internen Qualität möglichst effizient erreichen zu können, wird in Kapitel 3 das bekannte und für den Fokus dieser Arbeit relevante Konzept der *Softwaremaße* vorgestellt, da diese hervorragende Indikatoren für viele gewünschte Qualitätseigenschaften darstellen können. Für eine Übersicht wird eine allgemein übliche Klassifizierung bisher existierender Maße mit jeweiligen Beispielen vorgestellt, es werden verschiedene Anwendungsszenarien dargelegt, in denen Softwaremaße die Qualität verbessern helfen können, und es werden typische Vertreter von Werkzeugen aufgezeigt, die bei der

Vermessung hilfreich sind (die allerdings i.d.R. wiederum keinen Bezug zum erstellten Qualitätsmodell erlauben).
Trotz dieser im wissenschaftlichen Bereich durchaus etablierten Wissensbasis zum Thema Softwarevermessung wird diese immer noch nicht als Standardtechnik innerhalb der Softwareerstellung angesehen. Die wesentlichen Probleme des Einsatzes von Softwaremaßen zur Qualitätssicherung lassen sich dabei in zwei große Bereiche einordnen:

- Probleme, wie sie *grundsätzlich* bei jedem Einführen neuer Methoden und Werkzeuge innerhalb der Softwareentwicklung auftreten können: Für diesen Bereich existieren aus der Literatur bereits sieben Problemdimensionen, die sich, aufbauend auf eigenen Erfahrungen beim Einsatz von Maßen im Rahmen von Qualitätssicherungsmaßnahmen, direkt auf den Bereich der Softwarevermessung übertragen lassen und damit viele Praxisprobleme des Meßeinsatzes explizit und detailliert zu tage treten lassen.
- Probleme, wie sie *spezifisch* für den Bereich der meßbasierten Qualitätssicherung sind: Ein Großteil der zu diesem Bereich gehörenden Probleme, die in 11 wesentliche Problemdimensionen verfeinert werden können und deren Bearbeitung primärer Ansatzpunkt für diese Arbeit ist, haben direkte Auswirkungen auf die allgemeinen Probleme, so daß am Ende des Kapitels 3 die präzise Zielsetzung der Arbeit, die Verbesserung der internen Qualität eines Softwareprodukts durch das Anwenden des Konzepts der Softwarevermessung, deutlich heraus gearbeitet werden kann.

Das Ergebnis der dadurch motivierten Problemanalyse, das zum größten Teil auf eigenen Erfahrungen beim Durchführen von meßbasierten Qualitätssicherungen basiert, umfaßt zwei große Bereiche, die in Kapitel 4 und Kapitel 5 grundlegend behandelt werden:

1. *Fehlende Meßgrundlagen*: Zum einen verleitet das aus dem Alltag bekannte Konzept des Messens zu einer mangelnden Sorgfalt beim Transfer dieses Konzepts auf Softwareprodukte. Dies führt zu vielen Folgeproblemen wie z.B. mangelnde Akzeptanz in einer Entwicklungsabteilung oder Mißbrauch des Messens zur einseitigen Leistungsbewertung von Menschen.
2. *Schlechte Skalierbarkeit*: Zum anderen wird die Interpretation der Meßwerte mit zunehmender Menge vermessener Softwareteile immer schwieriger, d.h. das Meßkonzept skaliert nicht beliebig. Dies ist besonders gravierend, da die Notwendigkeit der Softwarevermessung mit der Größe des bzgl. der Qualität zu verbessernden Systems zunimmt.

1.2 Lösungsansätze in dieser Arbeit

Jede Qualitätssicherungsmaßnahme muß als ersten Schritt das gewünschte Ziel explizit identifizieren. Aufgrund der Vielschichtigkeit des Qualitätsverständnisses werden in Kapitel 2 die verschiedenen, existierenden Qualitätsmodelle der unterschiedlichen Qualitätsmodellarten analysiert, um deren gemeinsames Ziel heraus zu arbeiten. Für dieses wird eine einheitliche Nomenklatur vorgestellt, die es vereinfacht, die verschiedenen, existierenden Ansätze je nach Situation auszuwählen und durchaus auch in Mischformen anzuwenden, um als Ergebnis ein für einen konkreten Anwendungskontext präzises Bild der zu erreichenden Qualität zu liefern.
Diese Nomenklatur wird konsequent um den Bereich der Softwaremaße (Kapitel 3) erweitert. Speziell die Produktmaße werden, nach einer begründeten Kategorisierung, die hilft, einen groben Überblick über die Vielzahl existierender Produktmaße zu erhalten, in vier verschiedene, jeweils in eigenen Arbeiten empirisch evaluierte Prozesse eingebettet, so daß sie nicht nur als Technik, sondern als überprüfbarer, effizienter Prozeß zur Qualitätssicherung Anwendung finden können.
Da die Anwendung von Softwaremaßen aufgrund des Umfangs der vermessenen Objekte zwangsläufig die Werkzeugunterstützung fordert, wird in Kapitel 3 ebenfalls deren grundsätzlicher Aufbau vorgestellt und acht Dimensionen, mit denen die Evaluierung der mittlerweile zahlreich existierenden Meßwerkzeuge möglich ist, erarbeitet und für zwei typische Meßwerkzeuge beschrieben.

Für das Beheben des Problems der fehlenden Meßgrundlagen (s.o.) wird die häufig diametral zu den praktischen Meßproblemen liegende Meßtheorie verwendet, die im Kapitel 4 vorgestellt wird. Mit ihr ist es möglich, die wesentlichen, während des täglichen Vermessens meistens implizit angenommenen Bedingungen explizit zu machen. In vielen vorherigen Arbeiten ist dies jeweils nur in einer der beiden Extremformen vorgenommen worden:

- Die Meßtheorie wird als Einzeldisziplin betrachtet, innerhalb derer formale Anforderungskataloge erstellt werden, denen Maße entsprechen müssen. Der Schwerpunkt dieser Arbeiten liegt in der Meßtheorie selbst, so daß viele eher im praktischen Umfeld einer Messung begründeten Probleme nicht behandelt werden.
- Die Meßtheorie wird bei der Vermessung vollständig ignoriert, oder sie wird zwar vorgestellt, die Vermessung selbst wird aber völlig losgelöst davon durchgeführt, da die Meßtheorie nicht praxisnah genug scheint bzw. die Überprüfung der formalen Bedingungskataloge zu aufwendig erscheint.

Eine wesentliche in dieser Arbeit aufgezeigte Lösung ist die Beseitigung dieser separaten Betrachtungsweise von Meßtheorie und praktischen Meßproblemen. Das Explizieren wesentlicher Bedingungen für das Messen, die in dieser Arbeit primär innerhalb der empirisch erfahrbaren Welt gefordert werden, erlaubt u.a. für viele bisher angewendeten Softwaremaße eine deutliche Reduktion ihrer möglichen Aussagekraft. Darauf aufbauend werden sowohl für die Erstellung eigener als auch für die Validierung bekannter Maße jeweils eigene, wiederum praktisch erprobte Prozesse vorgestellt, so daß die Meßtheorie im Bereich der Softwarevermessung erstmals wirklich konstruktiv in der Praxis anwendbar wird.

Für das Beheben des Problems der schlechten Skalierbarkeit (s.o.) sind bisherige Ansätze nur ungenügend geeignet. Das wesentliche Problem ist die Interpretation einer Vielzahl von Meßdaten (da diese häufig identitätslos, d.h. losgelöst von der den Meßwert begründenden Software, dargestellt werden) einer Vielzahl von vermessenen Softwareteile (da diese häufig strukturlos, d.h. losgelöst von jeglichem Kontext des vermessenen Softwareteils, dargestellt werden). Das Ergebnis sind i.d.R. überlange Listen mit fragwürdigen Meßwerten, die weder eine bzgl. des jeweiligen Qualitätsmodells angepaßte Analyse, noch eine einfache Rückübertragung auf das den Meßwerten zugrundeliegende Softwaresystem ermöglichen. Der potentielle Nutzen von Softwaremaßen im Rahmen von Qualitätssicherungsmaßnahmen wird dadurch zunichte gemacht.

Für dieses Problem der Identifikation vieler Meßwerte von vielen vermessenen Softwareteilen wird in Kapitel 5 ein Modell erarbeitet, das zwei Aufgaben erfüllt: Auf der einen Seite erfaßt es die meisten in der Literatur aufgeführten und verwendeten Softwaremaße und zeigt deren gemeinsame Eigenschaften, und auf der anderen Seite wird ein wesentlicher, bisher noch nicht verwendeter Freiraum beim Vermessen motiviert, der im folgenden dieser Arbeit derart ausgenutzt wird, daß die meßbasierte Verbesserung der Qualität bestmöglich und vor allen Dingen auch für große Softwaresysteme unterstützt wird.

Für diesen Zweck wird in Kapitel 6 ein generisches, auf Ähnlichkeit beruhendes Distanzmaß vorgestellt, das vor allen Dingen drei Vorteile hat:

- Da es Anleihen aus der Psychologie und deren Modellen entnimmt, wie Daten im menschlichen Gedächtnis gespeichert und abgerufen werden, ist es sehr intuitiv, auch für größere Datenmengen anwendbar und zudem in Übereinstimmung zur Meßtheorie formulierbar (s.o. Problem der fehlenden Meßgrundlagen).
- Als echte *Metrik* erlaubt es das Anwenden von Techniken wie Clusteranalyse und Visualisierung in einer sehr direkten Art.

- Als echte Erweiterung des bisherigen Meßkonzepts von Software erlaubt es die erwartungsgemäß vollständige Integration bisheriger Maße.

Erste Einsätze des neuen Maßes werden in Kapitel 7 durch die Nachbildung einer speziellen Art bisheriger Maße demonstriert: die Kohäsionsmaße. Ihre Vielfalt – sowohl in der Art der Definition als auch der Art der betrachteten Kohäsionsart – wird dadurch klassifizierbar und mündet in Prozessen zur Erstellung eigener, dem jeweiligen Ziel bestmöglich angepaßter Instanziierungen des generischen Distanzmaßes.

Typische Instanziierungen werden in Kapitel 8 für objektorientierte Systeme vorgestellt. Diese umfassen sowohl konkrete Distanzmaße, die die bisher jeweils separat betrachteten Konzepte Kopplung und Kohäsion vereinen, als auch Distanzmaße, die auf allgemeinen Ähnlichkeit begründenden Merkmalen basieren

Um den praktischen Wert des generischen Distanzmaßes und dessen konkrete Instanziierungen in Kombination mit klassischen Softwaremaßen für den Zweck der Qualitätssicherung zu demonstrieren, werden in Kapitel 9 einige Fallstudien mit großen Softwaresystemen beschrieben, für die eine meßbasierte Qualitätsanalyse unter Verwendung der neuen Konzepte durchgeführt wurde. Dort werden ebenfalls die dafür notwendigen, auf klassischen Maßen basierenden Qualitätsmodelle vorgestellt.
Die Anwendung selbst wurde durch eigens dafür entwickelte Werkzeuge (*Crocodile* als Softwaremeßwerkzeug, *CrocoCosmos* als Softwarevisualisierungswerkzeug und ein Clusteranalyse-werkzeug) unterstützt. In allen Fällen haben die in dieser Arbeit vorgestellten Konzepte sehr deutlich ihren hohen Wert bei der Qualitätssicherung unter Beweis gestellt.

Im letzten Abschnitt werden – nach einer kurzen Zusammenstellung der wesentlichen Ergebnisse dieser Arbeit – einige Ausblicke formuliert, wie die hier vorgestellten Konzepte erweitert und auf andere Bereiche übertragen werden können.

1.3 Organisatorischer Kontext der Arbeit

Die vorliegende Arbeit ist als Dissertation während meiner fünfjährigen Mitarbeiterzeit am Lehrstuhl Software-Systemtechnik von Prof. Dr. C. Lewerentz an der Brandenburgischen Technischen Universität (BTU) Cottbus erarbeitet worden. Der thematische Schwerpunkt knüpft dabei an eine, ebenfalls von Prof. Dr. C. Lewerentz betreute, Dissertation von Karin Erni an, die die „*Anwendung multipler Metriken bei der Entwicklung objektorientierter Frameworks*“ [Erni96] erfolgreich innerhalb des ABB Forschungszentrums in Heidelberg durchführte.
Die bereits in Erni's Arbeit betonte Praxisnähe mit dem Ziel der Validierung der neu vorgestellten Konzepte und Werkzeuge ist in dieser Arbeit fortgesetzt worden: So sind die hier erarbeiteten neuen Ansätze in vielen Projekten, von denen nur einige wenige in Kapitel 9 erörtert werden, praktisch evaluiert und in Reflexionsphasen jeweils optimiert worden.
Die Notwendigkeit, den Anforderungen realer Projekte gerecht zu werden und sich eben nicht auf „akademische Systeme“ beschränken zu dürfen, führte zu einer sehr zuverlässigen Werkzeuglandschaft, die zudem als Public-Domain-Software von der Lehrstuhlseite bezogen werden kann (http://www.software-systemtechnik.de).
Grundlegende Ideen sind darüber hinaus mittlerweile als kommerzielles, ebenfalls am Lehrstuhl Software-Systemtechnik entwickeltes Produkt innerhalb der kommerziellen Software-Entwicklungsumgebung SNiFF+ (vgl. z.B. [Pfei97]) verfügbar.

„To many people,
quality is similar to what a federal judge
once commented about obscenity:
‚I know it when I see it‘ “
([Kan95], S. 2)

2 Softwarequalität

Der wirtschaftliche Erfolg eines Softwareprodukts hängt wesentlich mit der Einhaltung des *ökonomischen Prinzips* zusammen ([Wöhe96], S. 1f), das in der mengenmäßigen Definition zwei Formulierungen besitzt:

1. Ein vorgegebenes Ziel ist mit dem minimalen Ressourceneinsatz zu erreichen (*Minimalprinzip*), d.h. die für das Erreichen eines vorgegebenen Ziels notwendigen Ressourcen sollen minimiert werden.
2. Mit einem vorgegebenen Ressourceneinsatz ist ein maximales Ergebnis zu erzielen (*Maximalprinzip*), d.h. der Ertrag soll ohne Veränderung der notwendigen Ressourcen maximiert werden.

Bei der Betrachtung des Aufwands und des Ertrags entsprechend des ökonomischen Prinzips sind die Gesamtkosten über die vollständige Lebensdauer eines Produkts relevant (*total cost of ownership*, TCO). Dies ist besonders für den Bereich der Softwareprodukte von Bedeutung, da hier ein Großteil der Kosten erst während der Systemnutzung anfällt. Dieser sogenannte *Eisbergeffekt* (vgl. [Kahl98]) wurde durch viele Untersuchungen belegt (so z.B. 50-70% der Gesamtkosten einer Produkteinführung nach Auslieferung in [LiSw80]; neuere Zahlen belegen ähnliche Kostenverteilungen, vgl. z.B. [PuMy97], Kapitel 27).

Soll das ökonomische Prinzip für ein Softwareprodukt angewendet werden, folgt unmittelbar, daß die Systemnutzungskosten minimiert werden müssen. In der Softwarebearbeitung[1] werden Aktivitäten zur Modifikation des Produkts während der Systemnutzung häufig als *Wartung* bezeichnet und können verfeinert werden in (vgl. [PaSi94], [GhJaMa91] und [Pres97])

- *korrektive Wartung*, zu der das Beseitigen von Defekten bzgl. der Spezifikation oder das Einarbeiten erst während des Systemeinsatzes auftretender Anforderungen zählt,
- *adaptive Wartung*, die notwendige Änderungen am Produkt aufgrund geänderter Umgebungsparameter – wie z.B. durch Betriebssystemwechsel, Geschäftsprozessmodifikation oder neue Gesetzeslagen (z.B. ISO 9241 seit 1.1.2000, s.u.) – subsumiert, und
- *perfektive Wartung*, bei der das gegebene System bzgl. geforderter Eigenschaften verbessert wird (z.B. erhöhte Performanz, einfachere Bedienung, verbesserte Wartbarkeit).

Neuere Definitionen des Wartungsbegriffs umfassen ebenfalls die funktionale Erweiterung (*extensive Wartung*) und vorbeugende Veränderung (*präventive Wartung*) von Produkten (vgl. z.B. [Dumk00], S. 95ff).

Typisches Charakteristikum aller Wartungsarten ist, daß die konkreten Anforderungen für die jeweiligen Wartungsaktivitäten erst nach Produktfertigstellung vorliegen. Die Kosten einer Änderung nehmen aber überproportional mit dem Fertigstellungsgrad des Softwareprodukts zu (vgl. Abbildung 1).

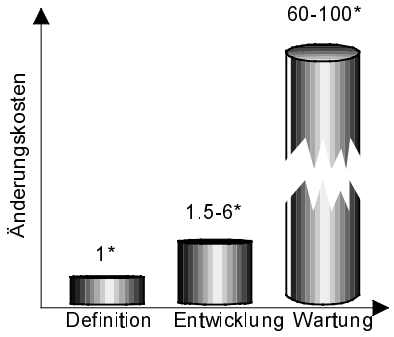


Abbildung 1: Änderungskosten vs. Produktfertigstellungsgrad (nach [Pres97], S. 18f)

[1] Unter *Softwarebearbeitung* werden in dieser Arbeit alle Aktivitäten an einem Produkt während des Software-Lebenszyklus verstanden, d.h. während „[...] *aller Phasen und Stadien dieser Produkte von ihrer Entwicklung, Einführung und Wartung bis zu ihrer Ablösung oder Beseitigung*“ ([Dumk00], S. 17).

Aus der Berücksichtigung der TCO und unter der Annahme, das ökonomische Prinzip anwenden zu wollen, folgt unmittelbar die Notwendigkeit, sowohl während der Softwareentwicklung als auch während der Softwarewartung Techniken anzuwenden, die eine spätere Wartung vereinfachen. Besitzt ein Softwareprodukt daraufhin solche Eigenschaften, so wird ihm *interne Qualität* zugesprochen (vgl. [GhJaMa91]):

> Ein Softwareprodukt besitzt *interne Qualität*, wenn es Eigenschaften besitzt, die es den Entwicklern ermöglichen, das Produkt mit kalkulierbarem Aufwand entsprechend gegebener Anforderungen zu warten.

Definition 1: Interne Qualität

Der Endbenutzer der Software, der direkt nur mit der *externen Qualität*, d.h. mit den nach außen sichtbaren Eigenschaften des Softwareprodukts, konfrontiert ist, profitiert von interner Qualität indirekt in Form geringerer Folgeversionskosten (*Update-Kosten*), schnellerer Fehlerbehebung, höherer Zuverlässigkeit des Produkts etc.
Externe und interne Qualität sind also eng miteinander verzahnt (vgl. [GhJaMa91]). Fenton bezeichnet diesen Zusammenhang als *Axiom des Software-Engineering* [Fent92].

Der allgemeine, übergeordnete Qualitätsbegriff, der interne und externe Qualität subsumiert, kann entsprechend des ISO-Standards definiert werden als:

> „*Qualität ist die Gesamtheit von Merkmalen einer Betrachtungseinheit bezüglich ihrer Eignung, festgelegte und vorausgesetzte Erfordernisse zu erfüllen.*"

Definition 2: Qualität allgemeinen (aus ISO 8402, Punkt 2.1 [ISO 8402]).

Die Vielschichtigkeit des Qualitätsbegriffs wird in dieser Definition durch die verschiedenen Arten von *festgelegten* und *vorausgesetzten Erfordernissen* ausgedrückt. Sowohl der Wunsch, das ökonomische Prinzip erreichen zu wollen als auch das damit verbundene Qualitätsverständnis sind nur eine mögliche, für die kommerzielle Softwareentwicklung aber sicher grundlegende Spezialisierung, die zudem häufig als grundlegende Forderung des *Software-Engineering* mit einfließt (vgl. z.B. [PaSi94], S. 48f und Kapitel 3).

Aufgrund der Vielzahl abhängiger Parameter des Begriffs Qualität (sowohl interner, externer als auch allgemeiner) sind Verfeinerungen und eine diese abdeckende, einheitliche Nomenklatur sinnvoll.
Die Verfeinerungen selbst sind notwendig, um

- Qualität explizit machen zu können, d.h. sie z.B. für ein gegebenes Produkt mittels Maßen, die die jeweiligen Verfeinerungen quantitativ bestimmen, zu ermitteln (siehe auch Kapitel 3),
- ein Produkt qualitativ zu verbessern (siehe auch Kapitel 9),
- einen Prozeß zu erstellen bzw. zu optimieren, der Produktqualität ermöglicht, und
- um Qualitätsstandards zu setzen, deren Einhaltung für spezielle Anwendungsbereiche relevant ist.

Qualität kann somit eine weitere Dimension in der Festlegung von Software-Anforderungen (*Requirements*) darstellen, die die herkömmlichen funktionalen, zeitlichen und budgetbezogenen Anforderungen ergänzt.

Im Abschnitt 2.1 werden für diesen Zweck einige konkrete Begriffsverfeinerungen vorgestellt, die sich jeweils in Form eines Qualitätsmodells (s. ebd.) darstellen lassen. In den in dieser Arbeit beschriebenen Praxisprojekten konnten diese fixen Qualitätsmodelle allerdings nicht sinnvoll angewendet werden. Die dort erfahrene Notwendigkeit der kontextbezogenen Anpassung führt im Abschnitt 2.2 zur Identifikation verschiedener Einflußfaktoren, die auf eine

Begriffsverfeinerung wirken können. Im Abschnitt 2.3 führt deren Vielschichtigkeit weg von konkreten Qualitätsmodellen hin zu aus der Literatur bekannten Prozessen, die als Ergebnis ein spezifisches, angepaßtes Qualitätsmodell haben. Allerdings zeigen auch diese in der Praxis deutliche Schwächen. So werden in Abschnitt 2.3.2 die bis dahin aufgeführten Möglichkeiten der Erstellung eines spezifischen, angepaßten Qualitätsverständnisses mit eigenen Erfahrungen kontrastiert und darauf aufbauend alternative Lösungen, deren gemeinsames Charakteristikum die größtmögliche Anpaßbarkeit zur kontextbezogenen Festlegung eines Qualitätsverständnisses ist, vorgeschlagen.

2.1 Fixe Qualitätsmodelle

Die Allgemeinheit des Begriffs Qualität und die damit häufig verbundene Unschärfe haben bereits früh zu Arbeiten geführt, die als Ziel die Verfeinerung des Qualitätsbegriffs hatten. Dieses geschieht durch sogenannte *Qualitätsmerkmale*, die ihrerseits weiter über mehrere Teilqualitätsmerkmale verfeinert werden können (vgl. [LRRA98]):

> Ein *Qualitätsmerkmal* ist ein Charakteristikum, dessen Ausprägung Einfluß auf übergeordnete Qualitätsmerkmale und damit – direkt oder indirekt – auf die Qualität besitzt.

Definition 3 : Qualitätsmerkmale

Diese Verfeinerungen von Qualität werden häufig in Qualitätsmodellen festgehalten:

> „*Ein Qualitätsmodell operationalisiert den allgemeinen Qualitätsbegriff mit Hilfe von Qualitätsmerkmalen.*“

Definition 4 : Qualitätsmodell (nach [Balz98], S. 257)

Diese Verfeinerung wird innerhalb der fixen Qualitätsmodelle so lange betrieben, bis die Qualitätsmerkmale mittels Maßen, Checklisten oder anderer Überprüfungstechniken erfaßbar sind. Gilb fordert darüber hinaus, daß zur Definition eines Qualitätsmodells ebenfalls für jedes Qualitätsmerkmal ein Meßkonzept, ein konkretes Meßwerkzeug, der schlechteste akzeptierte Wert, der geplante Wert, der bestmögliche Wert und – falls vorhanden – der aktuelle Wert gehört [Gilb87].
Aufgrund der häufigen Forderung, die Qualitätsmerkmale sollten untereinander minimale bzw. keine Überschneidungen besitzen (vgl. [ISO 9126]), werden Qualitätsmodelle häufig als Baum beschrieben und dargestellt. Ein bekanntes, der ISO 9126 zugrundeliegendes und vollständig als Baum modelliertes Qualitätsmodell ist in Abbildung 2 dargestellt.

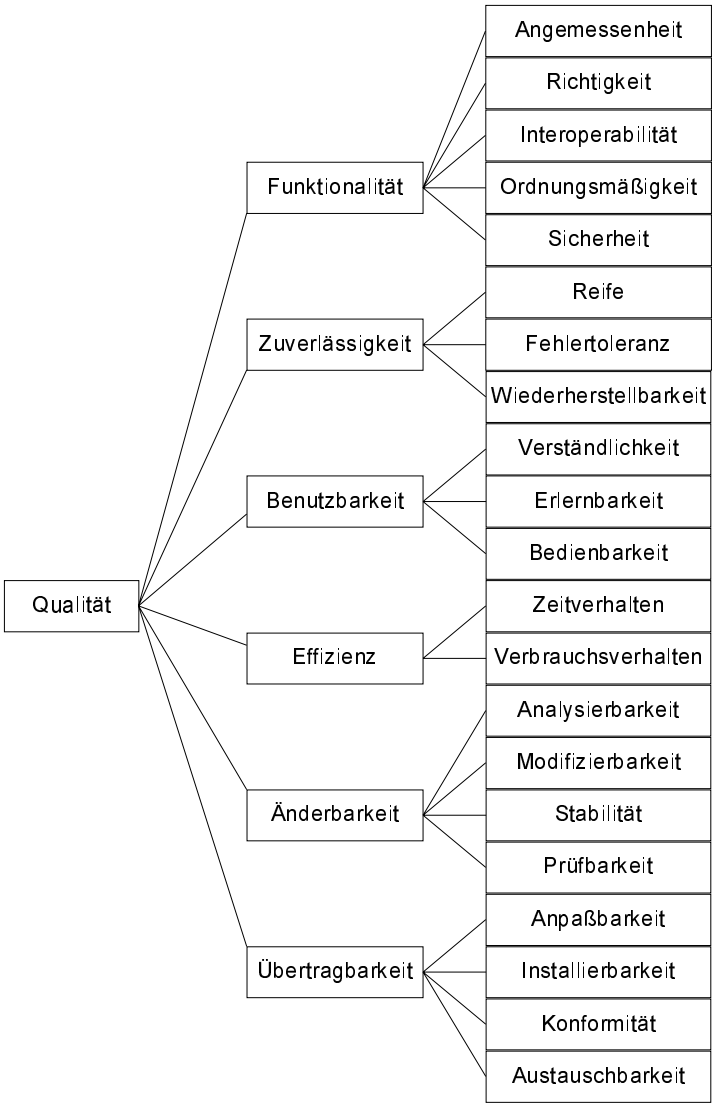


Abbildung 2: Qualitätsmodell entsprechend ISO 9126

Neben diesem standardisierten Qualitätsmodell gibt es viele weitere Qualitätsmodelle, die sich ebenfalls einer „Top-Down"-Verfeinerung des Qualitätsbegriffs bedienen (vgl. z.B. [Bowe76], [McRiWa77], [Boeh. et al. 78], „*DGQ-Modell*" in [DGQ86], „*CUPRIMDSO*" von IBM in [Kan95] oder „*FURPS*" von Hewlett Packard in [GrCa87]).
Qualitätsmodelle, deren Verfeinerungen mit dem Ziel vorgenommen werden, Qualität meßbar zu machen, zeichnen sich dadurch aus, daß bei den detailliertesten Verfeinerungen an die Stelle der Qualitätsmerkmale konkrete Maße formuliert werden, die das übergeordnete Qualitätsmerkmal quantitativ beschreiben. Solche Modelle werden häufig als *Factor-Criteria-Metrics-Model* (kurz: FCM-Modell) bezeichnet [McRiWa77]. Bei den Qualitätsmerkmalen wird hierbei zwischen *Faktoren*, von denen die Qualität direkt abhängt und die noch relativ abstrakt sind, und *Kriterien*, die die jeweiligen Faktoren weiter verfeinern, unterschieden. Entsprechend Abbildung 2 würde z.B. das Qualitätsmerkmal Änderbarkeit als Faktor und das Qualitätsmerkmal Modifizierbarkeit als ein diesen Faktor verfeinerndes Kriterium aufgefaßt werden. Maße sind in Abbildung 2 nicht mit eingezeichnet, da sie nicht Teil der ISO 9126 sind (vgl. Kapitel 3).

Um dem Ziel, das mit der Erstellung eines Qualitätsmodells verfolgt wird, gerecht zu werden, bedarf es für die einzelnen Qualitätsmerkmale einer präzisen Definition. Hinsichtlich des

Themenschwerpunkts dieser Arbeit sind vor allem die Begriffe *Änderbarkeit* (engl.: „Maintenance", s. „nationale Anmerkung" in Punkt 4.5 in [ISO9126]) und *Übertragbarkeit* (engl.: „Portability") interessant:

> *Änderbarkeit*: „*Eine Menge von Merkmalen, die sich beziehen auf den Aufwand, der zur Durchführung vorgegebener Änderungen notwendig ist.*"

Definition 5: Änderbarkeit (aus ISO 9126, Punkt 4.5 [ISO9126])

> *Übertragbarkeit*: „*Eine Menge von Merkmalen, die sich beziehen auf die Eignung der Software, von einer Umgebung in eine andere übertragen zu werden.*"

Definition 6: Übertragbarkeit (aus ISO 9126, Punkt 4.6 [ISO9126])

Bereits hier wird ersichtlich, daß diese Form von Qualitätsmodellen sehr diskutabel ist: während Wartbarkeit, wie sie weiter oben erläutert ist, die Portabilität als adaptive Wartung subsumiert, wird sie innerhalb der ISO 9126 als von der Wartung separat zu betrachtendes Qualitätsmerkmal aufgeführt.
Häufigstes Problem bei der Anwendung fixer Qualitätsmodelle ist die fehlende Möglichkeit, entsprechend spezifischer Parameter angepaßt zu werden (vgl. [SiRuLe00]). Qualität ist ein so vielschichtiger Begriff, daß es unmöglich ist, die verschiedenen Sichten in einem fixen Modell zu vereinigen. Darüber hinaus existieren bei der Verwendung fixer Qualitätsmodelle folgende Probleme:

- Die häufig verwendete Baumtopologie scheint zu restriktiv: Schon auf der Ebene verfeinerter Qualitätsmerkmale scheint ein 1:n Verhältnis zwischen Qualitätsmerkmal und Subqualitätsmerkmal unbegründbar. So ist es fragwürdig, ob die Anpaßbarkeit wirklich nur als Verfeinerung der Übertragbarkeit anzusehen ist, oder ob der Grad der Anpaßbarkeit nicht auch Auswirkungen auf die Änderbarkeit hat (vgl. Abbildung 2). Dies gilt erst recht für die Ebene der Maße. Die Lösung wäre ein allgemeines m:n-Verhältnis zwischen Qualitätsmerkmalen und Subqualitätsmerkmalen. Weitere Beispiele solcher m:n-Verhältnisse sind bei Dromey zu finden ([Drom95], S. 148f).
- Die häufig verwendete Verfeinerung in drei Stufen (Qualität ist abhängig von Faktoren, die wiederum von Kriterien bestimmt werden, die durch Maße quantifiziert werden können) scheint zu restriktiv, und ist ebenfalls nicht begründbar. Je präziser die Operationalisierung mittels Qualitätsmerkmalen geschieht, desto breiter und tiefer kann und soll das Qualitätsmodell werden. Diese Präzisierung ist als Gewinn und nicht als Modellverletzung zu interpretieren. So würde z.B. eine in das ISO-Qualitätsmodell durchgeführte Einflechtung des in der Literatur etablierten Wartungsbegriffs und dessen Verfeinerung in korrektive, adaptive und perfektive Wartung zumindest eine Verfeinerungstiefe von 4 nahelegen. In einem solchen Fall würde statt von Kriterium und Faktor, deren Unterschied weniger semantisch als modellbedingt anzusehen ist, auch allgemein nur noch von Qualitätsmerkmalen gesprochen werden.
- Eine mögliche Verfeinerung bis auf die Ebene von Maßen, Checklisten oder anderer Überprüfungstechniken genügt häufig noch nicht, da zur jeweiligen Überprüfungstechnik gleichzeitig eine Interpretationsvorschrift gehört, die angibt, welche Bereiche als qualitätsfördernd und welche als qualitätsvermindernd angesehen werden. Die oben genannten Forderungen von Gilb zielen ebenfalls in diese Richtung, sind aber in keinem verbreiteten fixen Qualitätsmodell erfüllt. In [DuFo97] wird für diesen Themenbereich der Begriff des *Qualitätsagenten* vorgestellt, der jeweils kontextspezifische Angaben zu den verwendeten Maßen, zu der jeweils vorzunehmenden Interpretation (d.h. zum Einfluß auf übergeordnete Qualitätsmerkmale), zu der Möglichkeit der Verfolgung der vermessenen Softwareteile über verschiedene Phasen der Softwareerstellung hinweg und Angaben zur Art der Präsentation der ermittelten Werte enthält ([DuFo97], S. 23).

- Die gegenseitige Abhängigkeit mancher Qualitätsmerkmale kann nicht mit modelliert werden, obwohl die Optimierung eines konkreten Qualitätsmerkmals häufig einen negativen oder positiven Einfluß auf ein anderes Qualitätsmerkmal ausübt. Allgemein können zwei Qualitätsmerkmale QMm_1 und QMm_2 drei verschiedene Beziehungen untereinander haben ([Wall90], [Gill92]):
  - *konkurrierend*, d.h. die Verschlechterung/Verbesserung von QMm_1 führt zu einer Verbesserung/Verschlechterung von QMm_2,
  - *verstärkend*, d.h. der Trend von QMm_1 führt zum selben Trend bei QMm_2 und
  - *indifferent*, d.h. QMm_1 und QMm_2 sind voneinander unabhängig.

  Ein Beispiel für zwei konkurrierende Qualitätsmerkmale ergibt sich aus dem Zusammenhang, daß eine Verbesserung der Wiederherstellbarkeit häufig eine Verschlechterung des Verbrauchsverhaltens nach sich zieht, da Daten, die potentiell wieder herzustellen sind, extra gehalten werden müssen.

  Weitere Beispiele für derartige gegenseitige Abhängigkeiten zwischen Qualitätsmerkmalen sind z.B. von Perry in Matrixform aufgezeigt [Perr87]: jeweils zwei Qualitätsmerkmale können dabei jeweils in einer der drei oben aufgeführten Beziehungen stehen; es wird dort allerdings nicht berücksichtigt, daß auch diese Abhängigkeiten selbst wieder von unterschiedlichen Einflußfaktoren (s.u.) abhängig sein können (vgl. [Gill92], Kapitel 2.3).

Dennoch können fixe Qualitätsmodelle viele konstruktive Hinweise beim Umgang mit Qualität liefern:

- *Nomenklatur*: Die allen Qualitätsmodellen beigefügten Begriffsdefinitionen können als einheitliche Nomenklatur innerhalb eines Projekts, einer Abteilung oder einer Firma dienen. Nur auf Grundlage eines einheitlichen Verständnisses über bestimmte Qualitätsmerkmale läßt sich Qualität überhaupt erst fassen (vgl. Kapitel 4).
- *Modellvorlage*: Ein fixes Qualitätsmodell kann als Ausgangsbasis für eigene Modifikationen herangezogen werden. Dromey z.B. verwendet das ISO-9126-Modell, erweitert es allerdings um das aus seiner Sicht sehr wichtige Qualitätsmerkmal "Wiederverwendbarkeit", das bei ihm wiederum – entsprechend des FCM-Ansatzes – von den Kriterien "Verwendung von standardisierten Sprachmerkmalen", "Vermeidung von Maschinenabhängigkeiten", "Implementierung einer wohldefinierten und gekapselten Funktionalität" und "volle Parametrisierbarkeit" ([Drom95], S. 148f) abhängt.
- *Diskussionsgrundlage*: Allein die Ablehnung eines fixen Qualitätsmodells aufgrund der intensiven Beschäftigung mit ihm fördert das Nachdenken über Qualität. Diese Sensibilisierung für und Diskussion über Qualität, evtl. sogar mit verschiedenen Personenkreisen (Management, Entwickler usw.), können bereits genügen, um erste Schritte zur Qualitätsverbesserung der erstellten Produkte zu motivieren (vgl. auch [SiRuLe00]).

Die Vielfältigkeit vorgeschlagener Qualitätsmodelle und deren Probleme beim Einsatz hängen größtenteils von der Vielzahl der Faktoren ab, die Einfluß auf ein angepaßtes Qualitätsverständnis haben. Diese sollen im folgenden aufgezeigt und klassifiziert werden.

2.2 Einflußfaktoren für ein angepaßtes Qualitätsmodell

Die verschiedenen Einflußfaktoren, die die verschiedenen Qualitätsvorstellungen für ein und dasselbe Produkt begründen, werden häufig anhand der verschiedenen Rollen, die Personen bei dem Umgang mit dem Produkt einnehmen, klassifiziert. Auf gröbster Abstraktionsebene wird zwischen den folgenden drei Rollen unterschieden (vgl. z.B. [Gill92], [PaSi95], [EvMa87]):

- *Benutzer*: Er ist primär an der externen Qualität des Produkts interessiert (s.o.). Die von ihm geforderten Qualitätsmerkmale konzentrieren sich auf funktionale Eigenschaften des Produkts wie z.B. die Robustheit, die Korrektheit und andere Eigenschaften, die während des Betriebs des Produkts von außen beobachtet werden können. Der Benutzer betrachtet das System als *Black-Box*, d.h. der Aufbau, die Funktionsweise und die internen Abhängigkeiten innerhalb des Systems sind nur insofern interessant, wie sie von außen beobachtbar sind.
  Die ISO 9241 beschreibt ein eigenes Qualitätsmodell für den Benutzer (vgl. [Rich95]); es umfaßt sowohl Qualitätsmerkmale bzgl. der verwendeten Hardware (Teil 3-9 der ISO 9241) als auch typische Anforderungen an die funktionale Leistungsfähigkeit (Teil 10 der ISO 9241) des Softwareprodukts. Sie ist seit dem 1.1.2000 in nationales Recht gewandelt worden, d.h. jede innerhalb von Deutschland verwendete Software, deren Verwendung maßgeblich den Bildschirm als primäres Informationskanal verwendet (dort: *Bildschirmarbeitsplätze*), hat die dort geforderten, externen Qualitätsmerkmale zu erfüllen (EG-Richtlinie 90/270/EWG bzw. DIN-EN 29241, vgl. auch Kapitel 3 in [Simo96]).
- *Entwickler*: Er ist primär an der internen Qualität des Produkts interessiert (s.o.). Die von ihm geforderten Qualitätsmerkmale konzentrieren sich auf nicht-funktionale Eigenschaften des Produkts wie z.B. die Verständlichkeit des Quelltextes, die Wiederverwendbarkeit einzelner Komponenten, die Konformität zu Standards und andere Eigenschaften, die bei der (Fort-) Entwicklung des Quelltextes relevant sind. Der Entwickler betrachtet das System als *White-Box*, d.h. typische Anwendungsszenarien, funktionale Anforderungen oder Änderungswünsche sind nur insofern für ihn interessant, wie sie Auswirkungen auf interne Details des Produkts haben.
- *Management* (innerhalb des die Software erstellenden Unternehmens): Es ist primär an der Einhaltung des ökonomischen Prinzips interessiert. Dazu gehört die Zufriedenheit der Benutzer, d.h. die Einhaltung dafür notwendiger Qualitätsmerkmale und eine hohe interne Qualität, um geforderte Wartungsaktivitäten mit kalkulierbaren und vertretbaren Kosten durchführen zu können. Das Management betrachtet das System als *Grey-Box*, d.h. es ist an einer groben Black-Box-Sicht und einer groben White-Box-Sicht des Produkts interessiert und kümmert sich um deren Kompatibilität zueinander.

Die einzelnen Qualitätsmerkmale von Qualitätsmodellen, die diese unterschiedlichen Rollen jeweils unterstützen, sind nicht zwangsweise disjunkt: Eine gut wartbare Software (Whitebox-Sicht) kann aufgrund verkürzter Fehlerbehebungs- (*Bugfix*-) -zeiten eine erhöhte Zuverlässigkeit (Blackbox-Sicht) begründen. Dieses Zusammenspiel wiederum erhöht die Wahrscheinlichkeit der Einhaltung des ökonomischen Prinzips (Greybox-Sicht).

Die im Rahmen dieser Arbeit durchgeführten Projekte haben gezeigt, daß die spezifischen Qualitätsmodelle der drei Rollen Entwickler, Benutzer und Management darüber hinaus vom betrachteten Produkt selbst abhängen. Folgende, durch das Produkt gegebene Faktoren, die auf die jeweiligen Qualitätsmodelle wirken, konnten dabei identifiziert werden:

- *Systemart*: Das konkrete Qualitätsmodell hängt wesentlich von der Art des Systems ab. So sind von Entwicklerseite z.B. für einen Wegwerf-Prototypen andere Qualitätsmerkmale relevant als für ein langlebiges Framework (vgl. z.B. [ErLe96]).

- *Systemeinsatz*: Das konkrete Qualitätsmodell hängt wesentlich von der Art des Einsatzes der Anwendung ab. So sind z.B. aus der Managementsicht an ein Computerspiel andere qualitative Anforderungen zu stellen als an eine Kraftwerkssteuerung. Der konkrete Einsatz hat ebenfalls Auswirkungen auf das Qualitätsmodell des Entwicklers: Hierzu sind gesetzliche Vorgaben (einsatzabhängige Standards wie z.B. der von der *U.S. Nuclear Regulatory Commission* verwendete *NUREG CR-6463*, [Mada99]) sowie häufig marktentscheidende Zertifizierungen (z.B. ISO 9000 oder CMM, vgl. Kapitel 3) zu zählen. Capers Jones hat für seine Softwaremessungen 16 verschiedene Einsatzklassen aufgeführt, die von minimalen Qualitätsanforderungen (Programme für den eigenen Gebrauch) bis zu maximalen Qualitätsanforderungen (Programme, die unter Militärauftrag stehen) angeordnet sind [Jone93].
- *Systementwicklungsstufe*: Wie bereits weiter oben erläutert, sollte Qualität nicht erst am Ende des Entwicklungsprozesses, sondern bereits für alle Zwischenergebnisse während der Entwicklung betrachtet werden. Mit zunehmendem Bewußtsein, daß der Auftraggeber auch in bestimmte Phasen der Erstellung des Systems integriert werden soll und kann (vgl. z.B. dessen notwendige Partizipation innerhalb des *eXtreme Programming* [Beck00]), gilt dies ebenfalls für die Rolle der Benutzer, die damit partiell Eigenschaften der Rolle des Entwicklers übernehmen. So sind z.B. für das Lasten- und Pflichtenheft andere Qualitätsmerkmale relevant als für die Entwurfsdokumente. Darüber hinaus kann eine Entwicklungsstufe mehrere, jeweils separat bzgl. Qualität beurteilbare Produkte liefern (z.B. verschiedene Modelldiagramme der Definitionsphase).
- *Systementwicklungsumgebung*: Dieser Faktor hat primär auf das Qualitätsmodell des Entwicklers Einfluß. Er berücksichtigt
  - *verwendete Werkzeuge*, die in Form angebotener oder angeforderter Funktionalitäten Einfluß auf das Qualitätsmodell haben können (z.B. erfordert die Verwendung von *LINT* spezielle, das Layout des Quelltextes betreffende Qualitätsmerkmale, [Darw91]),
  - *verwendete Programmierkonzepte und -sprachen*, die aufgrund inhärenter Schwächen und Stärken bestimmte Qualitätsmerkmale notwendig bzw. redundant erscheinen lassen. So werden bzgl. des Speichermanagements bei der Verwendung der Programmiersprache C++ beispielsweise andere Qualitätsmerkmale gefordert als bei der Verwendung von JAVA, da letztere hierfür weitestgehend selbst verantwortlich ist,
  - *involvierte Entwickler*, die aufgrund erkannter eigener Schwächen und Stärken (z.B. während der individuellen Anwendung des *Personal Software Process*‘, vgl. [Hump89]) ihr Qualitätsmodell anpassen. So wird z.B. ein Entwickler, der weiß, daß er einen großen Teil seiner Entwicklungstätigkeiten mit dem Herstellen der syntaktischen Richtigkeit seiner eigenen Quelltexte verbringt, Qualitätsmerkmale fordern, die seine diesbezügliche „Schwäche“, d.h. die Probleme mit der Syntax einer Programmiersprache, weitestmöglich kompensieren (z.B. durch Anwendung geeigneter Programmierrichtlinien, die den Vorrat des syntaktisch Machbaren bewußt reduzieren und die eventuell sogar durch die verwendete Entwicklungsumgebung unterstützt werden).

Diese Faktoren, die in unterschiedlicher Ausprägung über das Produkt in Form der verschiedenen Sichtweisen auf das System Einfluß auf das Qualitätsmodell der beteiligten Personenrollen nehmen, können wie in Abbildung 3 dargestellt werden.

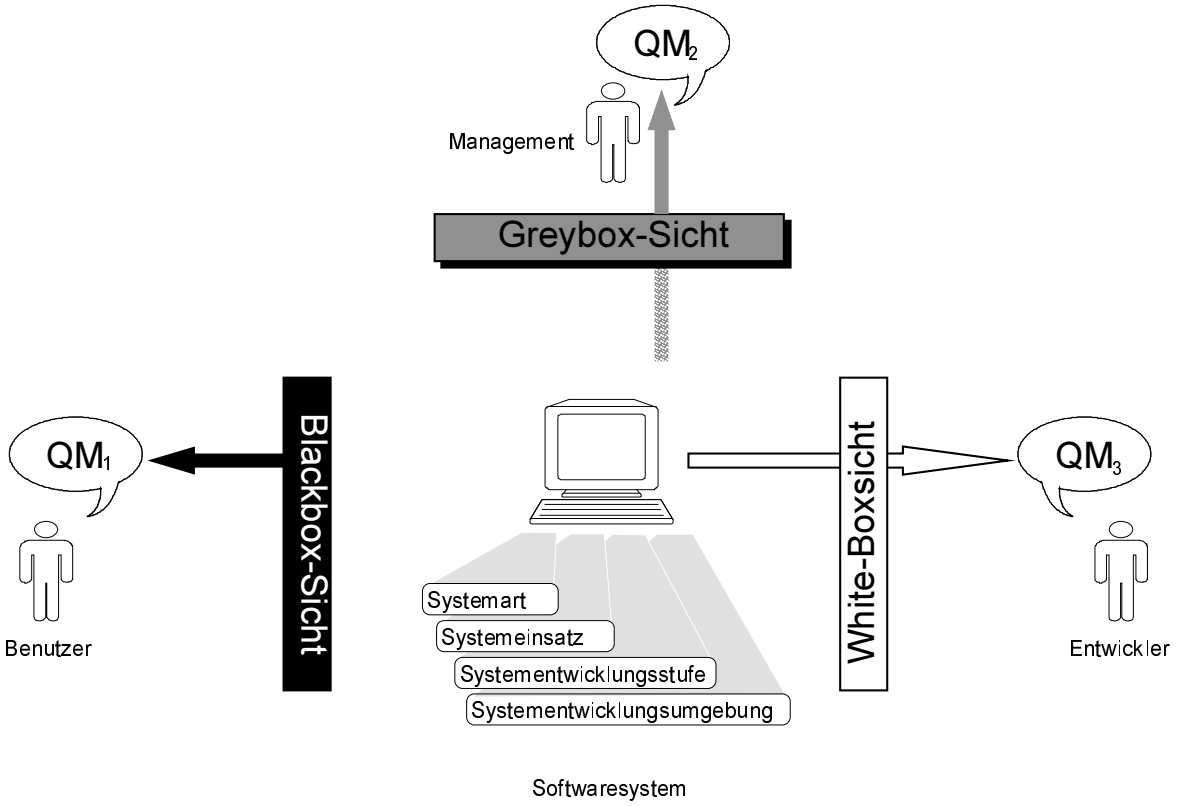


Abbildung 3: Verschiedene Rollen bei der Betrachtung eines Softwaresystems und deren unterschiedliche Qualitätsansichten

Die hier aufgezeigten Abhängigkeiten vom jeweiligen Qualitätsmodell können größtenteils auf das *multidimensionale Framework (MDF)* von Henderson-Sellers et al. übertragen werden, das die verschiedenen Dimensionen beschreibt, von denen die Auswahl geeigneter Maße abhängt [Hend96]. Dabei werden in der ersten Dimension die oben beschriebenen Rollen betrachtet, die zweite Dimension berücksichtigt Systementwicklungsumgebung und Systementwicklungsstufe und die dritte Dimension betrachtet die Granularität des betrachteten Produkts (in obiger Klassifizierung der Systementwicklungsstufe zugeordnet). In der bei Henderson-Sellers beschriebenen vierten Dimension werden die verwendbaren Maße betrachtet (s. Kapitel 3).

Die in diesem Abschnitt aufgezeigten verschiedenen Einflußfaktoren auf die Entwicklung eines adäquaten Qualitätsmodells machen deutlich, daß Qualitätsmodelle entweder sehr allgemein formuliert werden müssen, um allen Faktoren gerecht zu werden (und damit faktisch auch unbrauchbar, auf jeden Fall aber nur schwer überprüfbar werden), oder daß anstelle des Ergebnisses ein Prozeß beschrieben wird, wie ein adäquates Qualitätsmodell erstellt werden kann und wie dabei die verschiedenen Einflußfaktoren berücksichtigt werden müssen. Mit diesem Thema beschäftigt sich daher der nächste Abschnitt.

2.3 Prozesse zur Definition eines angepaßten Qualitätsmodells

In diesem Abschnitt werden mögliche Prozesse beschrieben, wie innerhalb einer konkreten Projektumgebung ein für den jeweiligen Kontext angepaßtes Qualitätsmodell erstellt werden kann. Selbst deren Anwendung ist weder ungeprüft auf andere Softwareprojekte übertragbar – da innerhalb einer anderen Projektumgebung die verschiedenen Faktoren wahrscheinlich anders einwirken – noch besitzt sie innerhalb ihres Entwicklungsprojekts zeitlose Gültigkeit. Letzteres ist einerseits mit der Dynamik einiger Einflußfaktoren begründbar (z.B. neue Werkzeuge oder

geänderte Anforderungen) als auch durch die Notwendigkeit gegeben, aus Fehlern in Systemen zu lernen und diese Erfahrungen in den Entwicklungsprozeß selbst wieder einfließen zu lassen. Je intensiver die Pflege dieses Regelkreises betrieben wird, desto höher wird der Prozeß z.B. innerhalb des CMM-Modells eingestuft (vgl. Kapitel 4).

2.3.1 Goal-Question-Metric-Methode

Die verschiedenen Prozesse zur Erstellung eines angepaßten Qualitätsmodells stellen i.d.R. jeweils lediglich Variationen der verbreiteten *Goal-Question-Metric*-Methode (GQM) dar [BaWe84], [BaRo88], [SoBe99]. Sie basiert wie die klassischen Qualitätsmodelle auf dem Prinzip der Verfeinerung, wobei drei Ebenen vorgeschlagen werden:

Goal: Diese Ebene behandelt die Frage, welche Ziele mit der Qualitätsbestimmung – im Fall der GQM-Methode mittels Messungen – verfolgt werden. Sie versucht dabei, das Einflußfaktorengerüst (vgl. Abbildung 3) zu konkretisieren, d.h. die Ziele in Abhängigkeit der Perspektive und der Umgebung zu beschreiben („*Goals are defined in terms of purpose, perspective and environment*", [BaRo88] S. 761).

Question: Auf dieser Ebene wird versucht, ein Ziel mittels verschiedener, quantifizierbarer Fragen zu definieren und quantitativ zu bestimmen („[...] *each goal generates a set of quantifiable questions that attempt to define and quantify the specific goal* [...]", [Basi95], S. 25). Diese sind dabei bereits auf die durch die Zieldefinition spezialisierte Sichtweise auszurichten. Die Schwerpunktsetzung auf quantifizierbare Fragen wird durch die Annahme gestützt, daß Messungen ein idealer Mechanismus zur Charakterisierung, Evaluierung und Vorhersage der verschiedenen Aspekte eines Softwareprodukts darstellen ([BaRo88], S. 759).

Metric: Auf dieser innerhalb dieses Modells feinsten Ebene werden diejenigen Maße bestimmt, die zur Beantwortung der Fragen beitragen sollen.

Die GQM-Methode selbst ist nicht auf die Qualitätsmodellbestimmung von Softwareprodukten beschränkt, sondern läßt sich allgemein auf ein vertieftes Verständnis von Prozessen, Ressourcen, Modellen und Maßen erweitern.
Als Beispiel dieser Methode ist in Abbildung 4 eine konkrete Instanziierung der GQM-Methode anhand eines Teil-Qualitätsmodells vorgestellt; dieses bezieht sich auf eine Qualitätsverbesserungsinitiative von Motorola für den Bereich der Softwareerstellung aus Managementsicht [Dask92].

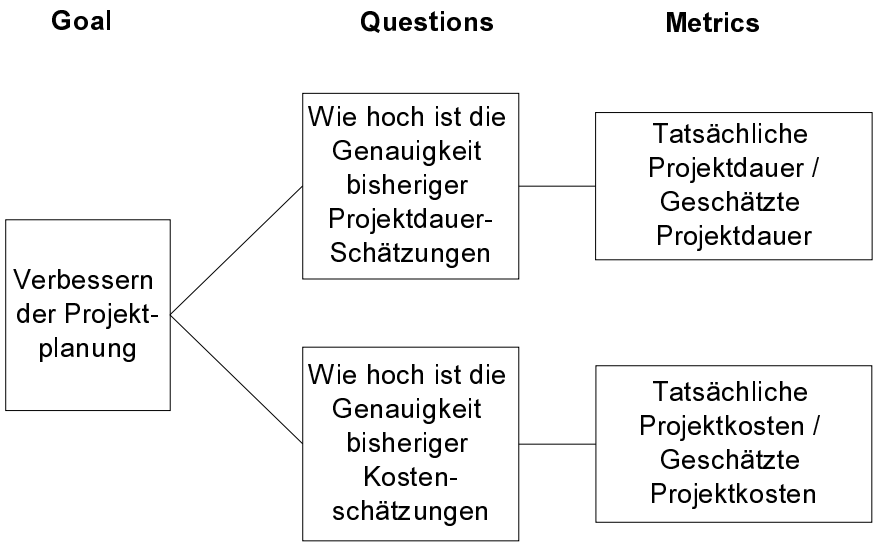


Abbildung 4: Beispiel einer Goal-Question-Metric-Instanziierung

2.3.2 Modifikationen der Goal-Question-Metric-Methode: Erfahrungen

Trotz der auf den ersten Blick bestechenden Einfachheit der GQM-Methode fallen bei der praktischen Anwendung innerhalb konkreter Softwareprojekte folgende Probleme negativ auf (vgl. [SiRuLe00]), die zum Teil bereits anhand der Beispielinstanziierung in Abbildung 4 deutlich werden und Anstoß für die Entwicklung modifizierter GQM-Prozesse waren:

- Die fixe Anzahl der 3 Verfeinerungsschichten ist für viele Qualitätsbereiche zu restriktiv. Wie bereits bei den fixen Qualitätsmodellen beschrieben, sollte eine detaillierte Verfeinerungskette, die sich über mehrere Abstraktionsschichten erstreckt, nicht als Modellverletzung, sondern als das Ergebnis einer sorgsamen Zielbestimmung angesehen werden. In der Praxis bewährt sich daher das Vorgehen, die Goals der GQM-Methode in hierarchisch angeordnete Subgoals zu verfeinern, die ihrerseits wiederum durch Questions und Subquestions hin zu Maßen verfeinert werden.
- Die einseitige Verfeinerung des Qualitätsbegriffs mittels des Top-Down-Ansatzes erschwert die Einbindung bereits existierender, genügend fein granularer Qualitätsmerkmale (vgl. [Drom95]). Diese sind z.B. in Form von Coding-Style-Guides, Dokumentationsrichtlinien oder anderen Projektrichtlinien gegeben. Das Fehlen des Bottom-up-Ansatzes innerhalb der GQM-Methode kann bestenfalls zur nachträglichen Identifikation von Qualitätsmerkmalen auf höheren Abstraktionsstufen führen, deren Verfeinerungen letztendlich das Einhängen der etablierten Qualitätsmerkmale ermöglichen. Schlechtestenfalls werden diese Richtlinien nicht in das angepaßte Qualitätsmodell eingebaut, so daß das resultierende Qualitätsmodell nicht vollständig ist und vermutlich eine eher managementorientierte als technikorientierte Sichtweise widerspiegelt.

  In der Praxis hat dieses Problem z.B. innerhalb des Unternehmens *Lloyd* zur Entwicklung eines umgekehrten GQM-Prozesses geführt, dem *MQG-Vorgehen* (metrics, questions, goals) [BaNe95]: Dort wird zuerst mittels verschiedener Maße gemessen, anschließend wird gefragt, warum die gemessenen Ergebnisse so aussehen und ob ein Potential zur Verbesserung erkennbar ist. Letztere werden dann als Ziele formuliert.
- Um das Qualitätsmodell auch während der Softwareerstellung möglichst effektiv einzusetzen, ist es hilfreich, die verschiedenen Qualitätsmerkmale, die größtenteils analytisch formuliert sind, um konstruktive Regeln zu erweitern, die dem Entwickler eine direkte Hilfe geben, wie das konkrete Qualitätsziel erreicht werden kann. Diese Erweiterung der GQM-Methode um die Ebene der Entwurfsregeln (*Design-Rules*) wird in [ErLe96] beschrieben. Auch wenn die direkte Einblendung dieser Ebene in das Qualitätsmodell das grundsätzliche Vorgehen der Verfeinerung verletzt und damit nicht direkter Bestandteil eines Qualitätsmodells sein sollte, so scheint doch der Nutzen einer expliziten Nennung von Konstruktionsrichtlinien zum Erreichen eines Qualitätsmerkmals äußerst hoch (vgl. [ebd.]). Die Angabe konstruktiver Regeln kann aber z.B. durchaus als weiteres Element der oben aufgeführten Qualitätsagenten aufgefaßt werden.
- Der klassische GQM-Ansatz läßt die Wichtigkeit der Kommunikation innerhalb der spezifischen Umgebung, für die ein Qualitätsmodell erstellt werden soll, nicht deutlich genug hervortreten. Selbst bekannte Begriffe sollten für jeden neuen Kontext einheitlich definiert werden, um Mißverständnissen vorzubeugen. So besitzt z.B. das Qualitätsmerkmal "Vollständigkeit" bei der Erstellung von Frameworks eine andere Bedeutung als bei der Entwicklung einer fertigen Stand-Alone-Applikation, da im ersteren Fall gerade die Unvollständigkeit bzgl. zwingend anzupassender Funktionalität deren Einsatzfähigkeit und deren Anpassungsfähigkeit erhöht. Ebenfalls wichtig ist die Einbeziehung sämtlicher Rollen in die Diskussion: Erst die Kenntnis der Wünsche und Probleme der Rollen untereinander ermöglicht den Konsens. Neuere Arbeiten zum GQM-Ansatz betonen zumindest die Vielfältigkeit möglicher Einflußfaktoren, in dem bei der Verfeinerung explizit die Dimensionen der *Sichtweise* (vgl. Rolle in Abschnitt 2.2), des *Anwendungsbereiches* (vgl. vom

Produkt abhängige Faktoren in Abschnitt 2.2), und des *Zwecks* (vgl. Kapitel 3.3) berücksichtigt werden sollen ([SoBe99], [Dumk00]).
Die Alternative zu diesen rollenübergreifenden Diskussionen sind jeweils zueinander disjunkte Qualitätsmodelle, die daraufhin entweder keine praktische Relevanz haben oder denen gemeinsam kein Produkt gerecht werden kann (vgl. [SiRuLe00]).

- Wie bereits bei der Diskussion um fixe Qualitätsmodelle aufgezeigt (vgl. Abschnitt 2.1), fehlt die explizite Benennung von Interpretationshilfen und die Darstellung von Abhängigkeiten zwischen Qualitätsmerkmalen und Maßen.
- Häufig sind die verschiedenen Verfeinerungen innerhalb eines spezifischen Kontextes unterschiedlich wichtig, d.h. es gibt weniger wichtige und besonders wichtige Qualitätsaspekte. Dromey regt daher z.B. an, die verschiedenen Verfeinerungen zu gewichten und nur diejenigen, die für den spezifischen Kontext besonders relevant sind, zu berücksichtigen, da anderenfalls z.B. jede Verfeinerung mit gutem Grund als Qualitätsmerkmal des Ziels "Wartbarkeit" betrachtet werden kann ([Drom95], S. 158).

Ein Großteil dieser Defizite sind auch in der jungen, und daher empirisch noch wenig explorierten GQM-Erweiterung GQM/MEDEA (GQM MEtric DEfinition Approach) von Briand, Morasca und Basili vorhanden [BrMoBa99]. Hier wird der Schwerpunkt auf die Definition theoretisch valider Maße gelegt, deren erwartete mathematische Eigenschaften bereits bei deren Definition als empirische Hypothese mit einfließen und somit Teile der Interpretationsvorschriften a priori festlegen.

In den im Rahmen dieser Arbeit durchgeführten Projekten hat sich gezeigt (vgl. Kapitel 8 und 9), daß weder die fertigen Qualitätsmodelle noch die fixen Prozesse eine problemlose Erstellung eines angepaßten, spezifischen Qualitätsmodells ermöglichen. Als sehr viel nützlicher wurde die bloße Konzentration auf den *Zweck* der Erstellung eines solchen Qualitätsmodells betrachtet. Die dadurch gewonnene Freiheit des Vorgehens, wie der Zweck zu erreichen ist, kann dann jeweils durch die kontextabhängige Wahl einer geeigneten (evtl. auch neuen) Technik (z.B. GQM, MQG oder Mischformen) unter der Prämisse größtmöglicher, dem Zweck dienlicher Anpaßbarkeit (z.B. beliebig viele Verfeinerungsebenen) ausgefüllt werden.
Unabhängig vom verwendeten Prozeß muß das erreichte Ergebnis immer eine möglichst kontextbezogene, eindeutige und vollständige Operationalisierung des verwendeten Qualitätsbegriffs mit Hilfe von Qualitätsmerkmalen darstellen. Je detaillierter die Qualitätsmerkmale sind, desto einfacher ist deren Bestimmung mittels Softwaremaßen (vgl. Kapitel 3). Ebenfalls identifizierte Abhängigkeiten zwischen einzelnen Qualitätsmerkmalen sind als zusätzliche Informationen zu dokumentieren. In der Praxis hat sich ebenfalls gezeigt, daß häufig allein die während des Prozesses der Erstellung eines solchen Qualitätsmodells stattfindende Diskussion innerhalb des Projekts großes Qualitätsverbesserungspotential für zukünftige Projekte birgt, da viele Diskussionsteilnehmer erst dadurch für den Qualitätsbegriff und dessen Vielschichtigkeit sensibilisiert werden.

Die nächste Aufgabe, nachdem ein angepaßtes, spezifisches Qualitätsmodell erstellt wurde, ist die Überprüfung, ob existierende oder in der Entwicklung befindliche Produkte den Anforderungen dieses Modells genügen. Zu diesem Zweck wird im nächsten Kapitel das Konzept des Messens eingeführt; allerdings ermöglicht erst das Qualitätsmodell die Entscheidung

- was gemessen werden soll,
- zu welchem Zweck gemessen werden soll,
- wie die Ergebnisse zu interpretieren sind und
- welche Konsequenzen daraus ableitbar sind.

Sämtliche Aktivitäten des Messens sollten immer jeweils im Kontext des sie motivierenden Umfelds gesehen werden.

2.4 Zusammenfassung

Die hohe Qualität von Produkten ist ein wichtiger Schlüssel für deren wirtschaftlichen Erfolg. Speziell für den Bereich der Software umfaßt das Eigenschaften, die Einfluß auf die Modifizierbarkeit des Produkts nach Auslieferung haben, da momentan hier die größten Kosten während des gesamten Software-Lebenszyklus anfallen. Die geringe Präzision einer solchen Forderung und die damit implizite Schwierigkeit der Überprüfung benötigt eine Präzisierung in Form eines Qualitätsmodells, in dem Qualitätsmerkmale hierarchisch angeordnet werden, so daß ein Abhängigkeitsgraph des allgemeinen Qualitätsbegriffs von einzelnen, leichter zu überprüfenden Qualitätsmerkmalen entsteht.
Bisherige Ansätze einer derartigen Festlegung von Qualität sind jedoch häufig in der Praxis nicht problemlos anwendbar:

- Die Verwendung von fixen Qualitätsmodellen scheitert häufig an der Multidimensionalität von Qualität, die sich vor allen Dingen in der Vielzahl von Einflußfaktoren manifestiert, die auf das jeweilige Qualitätsverständnis einwirken.
- Die für die Berücksichtigung der verschiedenen Rollen der an einem Softwareprodukt beteiligten Personen und deren individuelle Sichtweisen vorgestellten Prozesse zur Herleitung eines angepaßten, spezifischen Qualitätsmodells, sind häufig zu restriktiv in ihrem Vorgehen. Viele Probleme, die während der Qualitätsmodellerstellung damit in der Praxis häufig auftreten, bleiben unberücksichtigt.

An die Stelle konkreter Modelle oder Prozesse zur Modellerstellung rückt daher der allgemeine Zweck eines solchen Modells, der in unterschiedlicher Weise erfüllt werden kann. Erst die zweckdienliche Selektion, Mischung und Anpassung bisheriger Techniken erlaubt in der Praxis die erfolgreiche Erstellung eines angepaßten, spezifischen Qualitätsmodells als kontextbezogene, eindeutige und vollständige Operationalisierung des verwendeten Qualitätsbegriffs mit Hilfe von möglichst feingranularen Qualitätsmerkmalen.

Jegliche Bestimmung der Qualität anhand von Maßen kann erst nach einer solchen, einen Konsens suchenden Diskussion bzgl. des verwendeten Qualitätsverständnisses angegangen werden, da erst dann sinnvoll bestimmt werden kann, was zu welchem Zweck gemessen werden soll, wie Ergebnisse zu interpretieren sind und welche möglichen Konsequenzen daraus gezogen werden können.

„From a software engineering perspective, metrics
give you control of products and processes,
demonstrate the productivity, effectiveness, and quality of the work,
gain customer respect and credibility with management,
identify where improvements are needed,
let you know when to laugh and when to cry"
([GiDa95]: The benefits of metrics)

3 Softwaremaße

Nach der Konkretisierung des Begriffs der Qualität für ein spezifisches Umfeld über die Ebene der Qualitätsmerkmale hin zu Softwaremaßen ergibt sich die Möglichkeit, mittels dieser für das abhängige Merkmal jeweils einen Wert zu bestimmen, mit dem es für ein zu untersuchendes Objekt auftritt. Die möglichen graduellen Ausprägungen dieser Werte hängen dabei wesentlich vom zu bestimmenden Qualitätsmerkmal ab und reichen von binären Entscheidungen (z.B. *Test wird erfüllt* bzw. *Test wird nicht erfüllt*) bis hin zu beliebig komplexen Wertebereichen (z.B. ℚ für (*Anzahl entdeckter Fehler*) / (*Anzahl von Programmzeilen*)).
Die Bestimmung eines Wertes von einem Beobachtungsobjekt wird als *Messen*, die jeweilige Vorschrift, mit der einem Objekt dieser Wert zugeordnet wird, als *Maß* und der bestimmte Wert als *Meßwert* bezeichnet (für eine formalere Definition dieser Begriffe siehe Kapitel 4).

Das Messen im Umfeld der Softwarebearbeitung (vgl. Kapitel 2) ermöglicht das Bilden von *Feedback-Loops*, die es erlauben, auf Basis von gegenwärtigen Objektmeßwerten Optimierungsvorschläge für zukünftige Objektbearbeitungen herzuleiten. Eine allgemeine Form einer solchen Feedback-Loop ist mit dem *PDCA-Regelkreis* (Plan-Do-Check-Act) in Abbildung 5 dargestellt (vgl. [Iizu95], [YaKo95], [Kan95]).

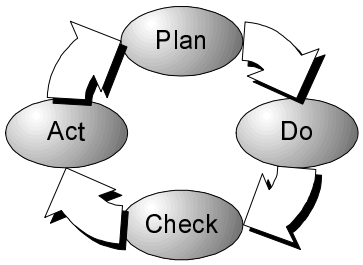


Abbildung 5: PDCA-Kreislauf

Die einzelnen Phasen bedeuten hierbei:

- *Plan*: Hier wird festgelegt, welche Objekte für eine Qualitätsbetrachtung im konkreten Kontext jeweils wie relevant sind (z.B. Entwurf, Implementierung, Dokumentation). Für diese werden entsprechend Kapitel 2 jeweils verfeinernde Qualitätsmerkmale bestimmt und deren Überprüfungsmöglichkeiten festgestellt. Das Ergebnis sollte ein angepaßtes Qualitätsmodell sein. Unter dessen Berücksichtigung wird die eigentliche Aufgabe des Prozesses spezifiziert.
- *Do*: In diesem Schritt werden die Aufgaben (z.B. Implementierung, Testen, Dokumentation) wie spezifiziert durchgeführt.
- *Check*: Hier wird der Ist-Zustand des Arbeitsprodukts mit dem in der Plan-Phase festgelegten Soll-Zustand verglichen.
- *Act*: Mögliche, während der Check-Phase aufgetauchte Abweichungen vom Soll-Zustand werden überarbeitet. Darüber hinaus wird versucht, die Ursache für die Abweichung zu ermitteln und in der nächsten Planungsphase zu berücksichtigen.

Andere Formen solcher Regelkreise – wie z.B. das *Quality Improvement Paradigm* oder das *Lean Enterprise Management* – lassen sich als detailliertere Formen des PDCA-Regelkreises auffassen (vgl. [Kan95]). Grundsätzlich stellen solche die Qualität berücksichtigende Prozesse allerdings weder ein hinreichendes, noch ein notwendiges Kriterium für die Erstellung eines qualitativ

hochwertigen Produkts dar: Notwendig nicht, da Qualität auch als akzidentelle Eigenschaft des Produkts präsent sein kann; hinreichend nicht, da die Güte der einzelnen Phasen von zahlreichen weiteren Parametern abhängt (z.B. Fachkenntnisse der Entwickler oder verwendete Entwicklungswerkzeuge). Die bewußte Verwendung eines solchen Prozesses bietet aber bei Produktproblemen die Möglichkeit der detaillierten Problemanalyse, d.h. an die Stelle des Zufalls, der zu akzidenteller Qualität führen kann, tritt der Fehler, der innerhalb des definierten Prozesses gemacht wurde; im Gegensatz zum Zufall ermöglichen Fehler allerdings das gezielte Lernen.
Messungen spielen speziell in der Check-Phase innerhalb des PDCA-Regelkreises eine herausragende Rolle, da sie – mit geeigneter Werkzeugunterstützung – den Ist-Zustand des Beobachtungsobjekts bzgl. gewählter Eigenschaften „objektiv" (vgl. Kapitel 4) und automatisch ermitteln können. Die Mächtigkeit von Messungen korreliert dabei unmittelbar mit der während der Planungsphase erreichten Verfeinerungstiefe: Je feingranularer die einzelnen Qualitätsmerkmale bestimmt wurden (evtl. sogar schon bis auf die Maßebene, vgl. GQM in Kapitel 2.3), desto einfacher ist deren Überprüfung mittels Maßen.
Die durch die Verwendung von Prozessen verbesserte Plan- und Kontrollierbarkeit innerhalb des Bereichs der Softwarebearbeitung begründete die Entwicklung weg von der ungeplanten ad-hoc-Durchführung einzelner Aufgaben (speziell für die Phase der Implementierung häufig als *hacken* bezeichnet) hin zu einer systematischen, disziplinierten und – für den Kontext dieser Arbeit besonders relevanten – quantifizierbaren Durchführung und mündete in der Entstehung des Wissenschaftsbereichs des *Software-Engineering*, das innerhalb der IEEE wie folgt definiert wird:

> „*The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is the application of engineering to software.*"

Definition 7: Software-Engineering (aus IEEE Standard Glossary 1990, in [Marc94], S. 1177)

In Deutschland hat sich für den Begriff des Software-Engineering die Übersetzung *Software-Technik* (gelegentlich auch in der orthographischen Variante „*Softwaretechnik*" verwendet, z.B. [HKLR84]) durchgesetzt (vgl. [Balz96], Seite 35ff).
Das Messen stellt ein großes Potential innerhalb der Software-Technik dar, deren Anforderungen (vgl. Definition 7) bei der Bearbeitung eines Softwareprodukts über dessen gesamte Lebensdauer gerecht zu werden.

In Abschnitt 3.1 werden verschiedene in der Literatur etablierte Klassen von Maßen, die für den Bereich der Softwarebearbeitung relevant sind, vorgestellt und mit klassischen Beispielen belegt. Der Schwerpunkt dieser Arbeit liegt dabei auf Maßen für die Softwareprodukte selbst und wird in Abschnitt 3.2. vertieft behandelt. Im Gegensatz zu anderen Arbeiten auf diesem Gebiet wird hier allerdings bewußt der Zusammenhang von Qualitätsmodell und den verwendeten Maßen hervorgehoben. Darauf aufbauend werden hier bereits Vorbedingungen des Messens heraus gearbeitet, deren unterschiedliche Ausprägungen zu einer hilfreichen Klassifizierung von Produktmaßen führen. Ein weiterer wesentlicher Schritt dieser Arbeit wird in Abschnitt 3.3 behandelt: Um Softwaremaße nicht bloß als eine weitere Technik vorzustellen, ist es unabdingbar, sie jeweils in Prozesse einzubetten. Diese müssen, damit sie effizient angewendet werden können, zwangsläufig durch entsprechende Werkzeuge bestmöglich unterstützt werden. Um speziell für die Produktvermessung die große Anzahl existierender Meßwerkzeuge einheitlich beschreiben und miteinander vergleichen zu können, werden in Abschnitt 3.4. acht Dimensionen vorgestellt, die als Basis jedes existierenden Produktmeßwerkzeugs dienen können. Dies wird in Abschnitt 3.5 exemplarisch für die zwei Meßwerkzeuge Datrix und Crocodile durchgeführt. In Abschnitt 3.6 werden, aufbauend auf den bis dahin vorgestellten Konzepten im Bereich der Softwarevermessung, die Probleme analysiert, die bis heute die breite Anwendung von Softwarevermessung behindern.

Dabei zeigt sich, daß viele der – größtenteils selbst erfahrenen – Probleme zu Problemklassen gehören, die typisch für jede Neueinführung neuer Techniken und Werkzeuge im Bereich der Softwarebearbeitung sind (Abschnitt 3.6.1). Deren Verfeinerung in spezifische Probleme des Messens (Abschnitt 3.6.2) erlaubt anschließend die detaillierte Formulierung der weiteren Zielsetzung dieser Arbeit (Abschnitt 3.6.3).

3.1 Klassifizierung

Softwaremaße werden primär nach der Art des vermessenen Objekts unterschieden. Da sowohl materielle Objekte (z.B. Personen), ideelle Objekte (z.B. Programme) als auch konzeptionelle Objekte (z.B. Prozesse) vermessen werden können, werden diese verschiedenen Arten häufig in den Begriff *Entität* subsumiert [FePf96].
Entsprechend der während der Softwarebearbeitung identifizierbaren, meßbaren Entitäten können Softwaremaße wie folgt klassifiziert werden [FePf96]:

- *Prozeßmaße* für diejenigen Aktivitäten, die sich auf die Bearbeitung von Software beziehen,
- *Ressourcenmaße* für diejenigen Mittel, die für einzelne Prozesse benötigt werden und
- *Produktmaße* für diejenigen Artefakte, die das Ergebnis eines Prozesses sind.

Jede Klasse von Softwaremaßen kann entsprechend der Abhängigkeit des betrachteten Merkmals einer zu vermessenden Entität von anderen Merkmalen oder Entitäten jeweils wiederum unterteilt werden in

- *direkte Maße*, d.h. das betrachtete Merkmal der zu vermessenden Entität besitzt keinerlei Abhängigkeit zu anderen Merkmalen oder Entitäten, und
- *indirekte Maße*, d.h. das betrachtete Merkmal der zu vermessenden Entität läßt sich nur indirekt über andere Merkmale und/oder Entitäten bestimmen.

Entsprechend dieser beiden Klassifizierungen ergeben sich sechs verschiedene Maßarten (vgl. Abbildung 6), von denen allerdings einige bzgl. der Vielfältigkeit deutlich eingeschränkt sind (schattierte Markierungen). Im folgenden werden für die Entitäten Prozesse und Ressourcen jeweils kurz typische Maße beschrieben, bevor in Abschnitt 3.2 der Bereich der Produktmaße vertieft behandelt wird.

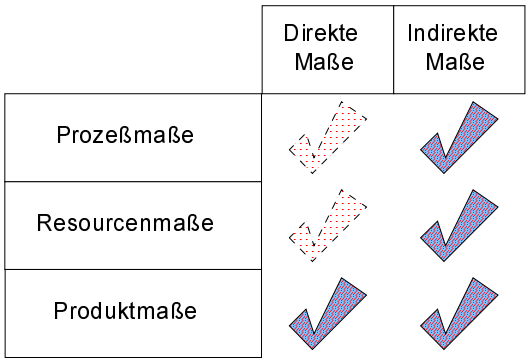


Abbildung 6: Softwaremaßklassifikation

3.1.1 Prozeßmaße

Jede Aktivität während der Softwarebearbeitung kann als Prozeß formuliert werden. Das Messen kann hier helfen, einen vertieften Einblick in die Effektivität des Prozesses zu bekommen. Da Prozesse und deren Effektivität häufig im Verhältnis zur aufgewendeten Zeit oder den angefallenen Kosten betrachtet werden, können die meisten Prozeßmerkmale nur indirekt gemessen werden (vgl. [FePf96]). Lediglich einfache Merkmale des Prozesses selbst, wie z.B. dessen Dauer oder das Vorkommen spezieller Ereignisse während des gesamten Prozesses, können direkt bestimmt werden.
Typische Beispiele für direkte und indirekte Prozeßmaße sind:

- Die Kosten einzelner Teilprozesse können direkt gemessen werden. Der prozentuale Anteil eines Teilprozesses an den Gesamtkosten dagegen kann nur im Verhältnis zu diesen bestimmt werden. Typische Prozeßmeßwerte bzgl. der Verteilung des Gesamtaufwands auf die einzelnen

Teilprozesse innerhalb des Wasserfallmodells haben z.B. bei Analyse von 125 Projekten innerhalb von *Hewlett-Packard* folgende Werte ergeben: 18% für Definition, 19% für Entwurf, 34% für die Implementierung und 29% für das Testen ([Grad92], S. 14).

- Vom *US Software Engineering Institute* (SEI) wurde für die US-Regierung ein Maß zur Beurteilung von Softwareunternehmen entwickelt, das messen soll, inwieweit die im Unternehmen verwendeten Prozesse die Erstellung von qualitativ hochwertiger Software unterstützen ([Hump89], [PfMc90]). Es handelt sich bei diesem *CMM-Modell* (Capability Maturity Modell) um ein fünfstufiges Modell. Das Modell selbst bestimmt die Prozeßgüte indirekt durch die Bestimmung unterschiedlicher, für jede CMM-Stufe typische Schlüsselaktivitäten (z.B. Stufe 1 durch ad-hoc-Aktionen, kaum prozeßgestützte Aktivitäten und fast ausschließlich Individualleistungen), die zusammen die jeweilige CMM-Stufe charakterisieren (sogenannte *key process areas, KPAs*). Diese Schlüsselaktivitäten werden im jeweiligen Unternehmen für dessen Einordnung einzeln bestimmt und anschließend mit Hilfe eines Schlüssels auf eine der 5 Ebenen abgebildet. Der erreichte Grad hängt dabei selbst wesentlich mit dem Einsatz von Maßen zusammen: So wird z.B. mit der vierten Stufe („*Managed*") die intensive Verwendung von Softwaremaßen und Qualitätsmodellen gefordert (vgl. [Kan95] und Kapitel 4.1.2).
  Das in Europa verwendete CMM Pendant *Bootstrap* stellt ebenfalls primär Prozeßmaße zur Verfügung; allerdings wird hierbei nicht die Güte des gesamten Prozesses auf einen Meßwert abgebildet, sondern es werden sogenannte *Prozeßgüteprofile* (*process maturity profile*) erstellt, d.h. Meßwerttupel unterschiedlicher Prozeßkriterien [KoMe95], die eine detailliertere Analyse unterstützen und mögliche Verbesserungspotentiale besser erkennen lassen.
- Die Effektivität eines Testprozesses wird häufig indirekt gemessen durch (*Anzahl gefundener Fehler*) / (*Dauer des Testprozesses*). Wird der Testprozeß selbst in aufwandsäquivalente Subprozesse aufgeteilt, so kann dieses Maß helfen, ein Abbruchkriterium zu definieren, ab wann ein Produkt ausgeliefert werden soll. Die Anzahl der im Produkt verbleibenden Fehler kann nicht als Kriterium herangezogen werden, da deren Anzahl i.d.R. unbekannt ist.
- Die Effektivität eines Einarbeitungsprozesses in ein fremdes System kann indirekt gemessen werden durch (*Anzahl betrachteter und grob verstandener Komponenten*) / (*Dauer der Einarbeitung*). Wird dieses Maß für mehrere Prozesse bestimmt, in denen fachlich vergleichbare Programmierer innerhalb einer vorgegebenen Zeit jeweils ein und dasselbe System verstehen lernen und dabei unterschiedliche Vorgehensweisen und Werkzeuge verwenden sollen, so kann damit eine Aussage über die Effektivität der verschiedenen Einarbeitungsvarianten gemacht werden. Auf diese Art und Weise wird z.B. in [SiRuKö99a] ein Vorgehen aufgezeigt, bei dem fremde Programme ohne die direkte Betrachtung des Quelltextes verstanden werden können. Da dessen Prozeßmaßergebnisse geringfügig besser waren als diejenigen, die für die sonst von den Probanden gewählten Verfahren gemessen wurden, konnte damit eine Verbesserung durch die neue Vorgehensweise belegt werden.

Die beiden letzten Beispiele belegen ein häufig wiederkehrendes Muster bei der Definition von Prozeßmaßen: Um die Güte des Prozesses bzgl. eines vorgegebenen Ziels (z.B. *Effektivität*) zu bestimmen, wird häufig die Güte des Prozeßergebnisses (z.B. *entdeckte Fehler* oder *betrachtete Komponenten*) ins Verhältnis zum benötigten Aufwand gesetzt. Produktmaße (vgl. Abschnitt 3.2) spielen daher auch für den Bereich der Prozeßmaße eine herausragende Rolle.
Das Messen von verwendeten Prozessen kann helfen, deren Effektivität zu erhöhen (z.B. durch Anwendung des PDCA-Regelkreises). Die Einrichtung solcher Feedback-Loops innerhalb eines Software bearbeitenden Unternehmens ist z.B. ein wichtiges Kriterium zum Erreichen höherer Ebenen innerhalb des CMM-Models.

3.1.2 Ressourcenmaße

Während Prozesse die Aktivitäten beschreiben, die ein Produkt als Ergebnis haben, werden unter Ressourcen diejenigen Entitäten verstanden, die zur Durchführung des Prozesses notwendig sind. Da diese Ressourcen die kostentreibenden Faktoren innerhalb der Softwarebearbeitung darstellen, werden deren Kosten ebenfalls häufig ins Verhältnis zum produzierten Ergebnis gesetzt. Diese allgemeine Form des Verhältnisses von Kosten zu Nutzen stellt eine zentrale Klasse innerhalb der Ressourcenmaße dar. Da sie immer mindestens jeweils von zwei betrachteten Eigenschaften abhängen, dominieren in der Klasse der Ressourcenmaße – wie bereits bei den Prozeßmaßen – die indirekten Messungen. Lediglich einfache Merkmale der Ressourcen selbst (z.B. *Kosten einer Entwicklungsumgebung* oder *Anzahl hinzu gekaufter Softwarebibliotheken*) lassen sich direkt bestimmen.

Typische Beispiele für direkte und indirekte Ressourcenmaße sind:

- Die Kosten einer Werkzeugumgebung für die professionelle Softwareentwicklung lassen sich direkt bestimmen. Deren Ökonomie läßt sich allerdings nur im Verhältnis zur dadurch eingesparten Entwicklungszeit oder der dadurch erhöhten Produktqualität bestimmen.
- Ein häufiges, für das Management interessantes Ressourcenmaß ist die Produktivität der einzelnen Softwareentwickler. Diese wird meistens durch das Verhältnis (*geschriebene Zeilen Programmtext*) / (*benötigte Zeit*) bestimmt. Um die Ergebnisse der Anwendung dieses Ressourcenmaßes vergleichbar zu gestalten, muß allerdings die implizite Annahme der Abstraktion von "Nutzen" auf "geschriebene Zeilen Programmtext" sorgfältig beachtet werden, da die Ergebnisse ansonsten z.B. durch überflüssige Programmtextzeilen verzerrt werden können.
- Die Zeit, die ein Computer für das Abarbeiten eines Programms benötigt, läßt sich allgemein als Verhältnis von der Anzahl abgearbeiteter Arbeitsschritte zur Zeit bestimmen. Normierungen bzgl. der Art der Arbeitsschritte ermöglichen den Geschwindigkeitsvergleich zwischen verschiedener Hardware. Verschiedene solcher *Benchmark-Messungen* (vgl. z.B. [Weic90]) stehen dabei für die verschiedenen Arten von Normierungen (z.B. Fokussierung auf ausgabelastige oder speicherintensive Arbeitsschritte).

Sowohl die Prozesse als auch die Ressourcen dienen der Fertigstellung eines Produkts. Für viele indirekte Prozeß- und Ressourcenmaße spielen daher Messungen bestimmter Merkmale des Produkts, die im folgenden Abschnitt erläutert werden, eine wesentliche Rolle.

3.2 Produktmaße

Produkte sind die Ergebnisse eines Prozesses, für dessen Durchführung Ressourcen benötigt werden. Als Produkte werden damit nicht nur Quelltext und ausführbare Programme bezeichnet, sondern ebenfalls Dokumentation, Spezifikation, Entwurf, Projektplan usw. Der Schwerpunkt dieser Arbeit liegt im Bereich der Quelltextvermessung, da die Quelltextqualität wesentlichen Einfluß auf den wirtschaftlichen Erfolg eines Unternehmens hat. Des weiteren liegen Quelltexte größtenteils bereits in elektronischer Form vor (z.B. in Dateien), was deren automatische Vermessung vereinfacht bzw. aufgrund der üblichen Produktgrößen die Vermessung erst möglich macht.

Im Bereich der Produktmaße werden sowohl direkte als auch indirekte Maße verwendet. Um die gemessenen Merkmale entsprechend dieser Maßklassifikation unterscheiden zu können, wird der Begriff des *internen* und *externen Merkmals* einer Entität eingeführt [FePf96]:

Ein *internes Merkmal* einer Entität besitzt keine Abhängigkeit zu anderen Merkmalen von derselben Entität oder von anderen Entitäten der für die Betrachtung relevanten Umgebung. Besteht dagegen mindestens eine derartige Abhängigkeit, wird das Merkmal als *externes Merkmal* bezeichnet.

Definition 8: Internes und externes Merkmal

Diese Definition erlaubt die – in der Wissenschaft bis jetzt häufig übersehene – einfache Integration des Konzepts des Qualitätsmodells (vgl. Kapitel 2) in den Bereich der Produktvermessung und beleuchtet noch einmal den eigentlichen Zweck von Qualitätsmodellen von einem anderen Standpunkt:
Die Bestimmung, ob ein zu betrachtendes Merkmal intern oder extern ist, kann diskutabel sein (vgl. Kapitel 4), jedoch wird ein Qualitätsmerkmal, das sich innerhalb des Qualitätsmodells noch auf einer sehr geringen Detaillierungsstufe befindet, von mehr anderen Merkmalen abhängen als ein Merkmal, das bereits die maximale Verfeinerung darstellt, da das Prinzip der Verfeinerung eines Qualitätsmerkmals in mehrere Subqualitätsmerkmale dort jeweils eine Konzentration auf einen Teilbereich und damit eine Reduktion der abhängigen Merkmale bewirkt. Mit dieser graduellen Unterscheidung der Merkmale ist es möglich, ein Qualitätsmodell als hierarchischen Abhängigkeitsgraph zwischen „externeren" und „interneren" Merkmalen aufzufassen. Die bereits hier ersichtliche Schwierigkeit bei der Bestimmung der Ausprägung von externen Merkmalen läßt sich direkt auf den Bereich der *indirekten Maße*, die externe Merkmale bestimmen sollen, übertragen (vgl. Kapitel 4). Je präziser das Qualitätsmodell, desto mehr *direkte Maße*, d.h. Maße, die interne Merkmale bestimmen, können zum Einsatz kommen, die das zu bestimmende Qualitätsmerkmal dann aufgrund dessen höherer Präzision sehr viel genauer bestimmen können.

Neben der Klassifizierung in interne und externe Merkmale werden im Bereich der Softwareprodukte (z.B. Quelltext) verschiedene Merkmalsklassen unterschieden, die jeweils einer bestimmten Sichtweise auf das Produkt entsprechen. Diese Sichtweise wird als *Produktmodell* bezeichnet:

Ein *Produktmodell* stellt ein Abbild des Produkts dar, welches Analogien zum Produkt aufweist und durch einen oder mehrere Aspekte eine Abstraktionsbeziehung zum Produkt besitzt. Das Ziel von Produktmodellen ist die Reduktion der Komplexität durch Weglassen von Eigenschaften und Berücksichtigung nur eines oder einiger weniger Aspekte.

Definition 9: Produktmodell (in Anlehnung an Modelldefinition in [Petr98], S. 10f)

Für Produkte lassen sich folgende Produktmodellklassen identifizieren, die ihrerseits jeweils eine bestimmte Merkmalsklasse repräsentieren[2]:

- *Größenmodelle*, die den Umfang eines (Teil-) Produkts in unterschiedlicher Weise modellieren und

[2] Die häufig aufgeführte Klasse der *Komplexitätsmodelle* wird hier nicht explizit aufgeführt, da es aufgrund der Vielschichtigkeit des Begriffs Komplexität kein Modell (und damit auch kein Maß) gibt (und auch nicht geben kann), welches Komplexität vollständig und universell modelliert (vgl. [Fent94]). So wie Qualität als vielschichtiges Konzept betrachtet werden muß (vgl. Kapitel 2), so muß auch Komplexität als eine *multidimensionale Eigenschaft* aufgefaßt werden, deren Bestimmung z.B. durch einen *Komplexitätsmeßvektor* möglich ist ([Eber95], S. 170ff). Der Vektor selbst besteht, ähnlich den Verfeinerungen von Qualität mittels Qualitätsmerkmalen, aus verschiedenen Teilmodellen (dort z.B. verfeinert mittels 16 Maßen), die jeweils durch obige Klassifizierung wieder vollständig abgedeckt sind (z.B. durch viele Größenmaße, vgl. [ebd.]).

- *Strukturmodelle*, die mehrere Teilprodukte und mögliche Relationen zwischen ihnen modellieren, und die ihrerseits verfeinert werden können in
  - *Kopplungsmodelle*, die explizit gemachte Verbindungen zwischen den Teilprodukten modellieren und
  - *Kohäsionsmodelle*, die verschiedene Formen des inneren Zusammenhalts von Teilprodukten modellieren.

Für jede dieser drei in dieser Arbeit heraus gearbeiteten Produktmodellklassen werden im folgenden jeweils kurz typische Merkmale dieser Klasse und entsprechende Maße vorgestellt, um zu belegen, daß diese Klassifizierung äußerst hilfreich beim Analysieren gegebener und neu zu erstellender Maße ist, da für die Interpretation relevante Gemeinsamkeiten und Unterschiede zwischen verschiedenen Maßen deutlich werden.

3.2.1 Größenmodellmaße

Das wohl älteste Maß für Softwareprodukte (und hier speziell für Quelltexte) ist die Bestimmung der Anzahl von Zeilen, aus denen das Produkt besteht (*lines of code, LOC*) (vgl. [DFKW96]). Der einfachsten Variante dieses Maßes liegt ein Produktmodell zugrunde, in dem jede Zeile des Quelltextes im Ausdruck als einfach zu zählender Punkt notiert wird (*Brutto-LOC, BLOC*). Für die Ausblendung der darin enthaltenen Leer- und Kommentarzeilen kann das Produktmodell dahingehend modifiziert werden, daß diese bei der Modellbildung ausgespart werden, so daß nur noch für die Programmausführung relevante Zeilen berücksichtigt werden (*Netto-LOC, NLOC*). Eine weitere Produktmodellmodifikation versucht, die in vielen Programmiersprachen mögliche Sequenz mehrerer Anweisungen innerhalb einer Zeile für das Größenmaß zu berücksichtigen, in dem jede einzelne Anweisung als zu zählender Punkt notiert wird. Empirische Untersuchungen haben ergeben, daß die Meßwerte des Maßes LOC je nach gewähltem Produktmodell bis um den Faktor 5 variieren können [Jone86].
Ein weiteres sehr verbreitetes Größenmodellmaß zur Bestimmung des Umfangs vor allem prozeduraler Produktteile ist *McCabe's zyklomatische Zahl* [MCCA76], die die Anzahl der im Produkt existierenden, linear unabhängigen und vollständigen Pfade bestimmt[3]. Das für die Bestimmung der zyklomatischen Zahl notwendige Produktmodell besteht in der Regel aus einem Kontrollflußgraphen, in dem die Knoten durch die existierenden Anweisungen und die gerichteten Kanten zwischen den Knoten jeweils durch mögliche Kontrollflüsse gegeben sind. Innerhalb eines solchen Kontrollflußgraphen ist die zyklomatische Zahl definiert als ((*Anzahl der Kanten*) – (*Anzahl der Knoten*) +2).
Einfache Größenmodellmaße für objektorientierte Systeme versuchen häufig, die Größe existierender Klassen zu bestimmen. Mögliche Modelle hierfür sind z.B. (vgl. [LoKi94])

- die Bestimmung der Anzahl der in einer Klasse definierten Methoden (*Number of Methods, NoM*),
- die Bestimmung der Anzahl der in einer Klasse definierten und nach außen sichtbaren Methoden (*Number of public Methods, NoPubM*),
- die Bestimmung der Anzahl der in einer Klasse definierten Attribute (*Number of Attributes, NoA*) oder
- die Summierung der LOC-Werte der in einer Klasse definierten Methoden (*Weighted Method Count, WMC*).

[3] Die zyklomatische Zahl bestimmt nicht – wie häufig beschrieben – die Anzahl aller möglichen Programmpfade; vielmehr transferiert McCabe Techniken der linearen Algebra auf Kontrollflußgraphen und bestimmt eine *Basis von Programmdurchläufen*, mittels derer alle möglichen Programmpfade als Linearkombinationen dargestellt werden können; die zyklomatische Zahl gibt lediglich an, aus wievielen linear unabhängigen Programmdurchläufen jede Basis für einen prozeduralen Produktteil besteht, vgl. [MCCA82].

3.2.2 *Kopplungsmodelle*

Innerhalb dieser Modellklasse liegt der Schwerpunkt der Betrachtung auf der expliziten Vernetztheit der Komponenten untereinander ([StMyCo74], [YoCo79]). Die Kopplungsmodelle unterscheiden sich primär durch den Grad der Produktabstraktion und der Art der Vernetzung zwischen Objekten dieser Abstraktion: So können sowohl einzelne Funktionen eines Programms, die miteinander z.B. durch eine Benutztbeziehung vernetzt sind, betrachtet werden, als auch ganze Programmpakete, die durch eine Instanziierungsbeziehung vernetzt sind (z.B. ein GUI-Framework und mehrere darauf aufbauende Anwendungen).

Ein frühes und weit verbreitetes Kopplungsmodell betrachtet die durch Informationsfluß zwischen Softwaremodulen entstehende Vernetzung [HeKa81]. Das Modell selbst besteht aus einem gerichteten Graphen, in dem die Knoten durch die zu betrachtenden Softwaremodule $\mathcal{M}_i$ und die innerhalb des Systems global definierten Variablen v_k bestimmt sind. Zwischen einem Modul $\mathcal{M}_a$ und einem Modul $\mathcal{M}_b$ existiert genau dann eine gerichtete Kante e_{call}, falls das Modul $\mathcal{M}_a$ das Modul $\mathcal{M}_b$ aufruft. Zwischen einem Modul $\mathcal{M}_a$ und einer Variablen v_b können sowohl gerichtete Kanten e_{read} existieren, wenn das Modul $\mathcal{M}_a$ die Variable v_b lesend benutzt, als auch gerichtete Kanten e_{write} vorkommen, wenn das Modul $\mathcal{M}_a$ die Variable v_b schreibend verwendet. Aufbauend auf diesem Kopplungsmodell definieren Henry und Kafura für ein Modul $\mathcal{M}$ die beiden Kopplungsmodellmaße [ebd.]

- *Fan-in($\mathcal{M}$)*, das sich berechnet aus der Anzahl ankommender Kanten e_{call} plus der Anzahl ausgehender Kanten e_{read}, und
- *Fan-out($\mathcal{M}$)*, das sich berechnet aus der Anzahl ausgehender Kanten e_{call} plus der Anzahl ausgehender Kanten e_{write}.

Mittels dieser Maße ist es möglich, den intermodularen Informationsfluß zu bestimmen, wobei sowohl Kopplung zwischen Modulen als auch zwischen Modulen und globalen Variablen betrachtet wird.

Diese Kopplungsmodelle lassen sich direkt auf den objektorientierten Bereich übertragen, wobei als Module i.d.R. Klassen, Subsysteme (Gruppe von Klassen) und Methoden angesehen und als Variablen die durch eine Klasse definierten Attribute betrachtet werden. Das bekannteste Kopplungsmaß für objektorientierte Systeme *CBO* (*coupling between objects*, [ChKe94]) betrachtet die Vernetztheit der Klassen untereinander, die dort durch Benutzungsbeziehungen gegeben ist: Eine Klasse C_i benutzt dabei eine Klasse C_j, wenn mindestens eine Methode der Klasse C_i mindestens eine Methode oder eine Instanzvariable der Klasse C_j benutzt. Der CBO-Wert einer Klasse C_i ist durch die Anzahl unterschiedlicher Klassen C_j bestimmt ($i \neq j$), die von C_i benutzt werden.

Innerhalb einer weiteren Form von Kopplung betrachtenden Produktmodellen ist die Vernetztheit der Komponenten untereinander durch eine Vererbungsbeziehung gegeben. Maße, die auf diesem Produktmodell basieren, sind z.B.

- *NoC* (*Number of Children*, [ChKe94]), das für eine Klasse die Anzahl direkter Unterklassen angibt,
- *NoP* (*Number of Parents*), das für eine Klasse die Anzahl direkter Oberklassen angibt und
- *DIT* (*Depth of Inheritance Tree*, [ChKe94]), das für eine Klasse den längsten Pfad von der Klasse im Vererbungsbaum bis zu der Wurzel angibt.

3.2.3 Kohäsionsmodelle

Innerhalb dieser Modellklasse liegt der Schwerpunkt der Betrachtung auf dem Grad der Zusammengehörigkeit von Elementen einer übergeordneten Komponente [StMyCo74]. Die Kohäsionsmodelle unterscheiden sich primär durch den Grad der Produktabstraktion und der Art der Zusammengehörigkeit. So können sowohl einzelne Anweisungen innerhalb einer Funktion zusammengehören und eine spezielle Aufgabe implementieren als auch ganze Teilprogramme zusammengehören, die eine gemeinsame Aufgabe bewältigen (vgl. [ebd.]). Für eine weiterführende allgemeine Betrachtung von Kohäsion vgl. Kapitel 7; eine Spezialisierung für objektorientierte Systeme erfolgt in Kapitel 8.

Eines der frühesten Kohäsionsmodelle basiert auf einem Produktmodell, das nicht automatisch aus dem zugrundeliegenden Produkt gewonnen werden kann [StMyCo74]: Die umgangssprachliche, auf einen Satz beschränkte Funktionsbeschreibung, die vom Entwickler für jede geschriebene Funktion notiert werden muß. Für dieses Kohäsionsmodell wird ein Maß angegeben, das die Funktionsbeschreibung auf sechs geordnete Kohäsionsgrade abbildet, die von *zufällig*, d.h. keine Kohäsion, bis hin zu *funktional*, d.h. maximale Kohäsion, reichen (vgl. Kapitel 7). So wird z.B. die Funktionsbeschreibung bzgl. zeitlicher Angaben überprüft (gegeben durch temporale Adverbien wie „zuerst", „danach" oder „wenn"). Ist diese Überprüfung positiv, so liegt eine temporäre Kohäsion vor, d.h. die Elemente in dem betrachteten Modul gehören sowohl logisch als auch bzgl. der Abarbeitungsreihenfolge zusammen. Diese Form des Vermessens kann allerdings nur bedingt automatisiert werden und ist aufgrund der notwendigen manuellen Produktmodellerstellung kaum praxisrelevant.

Ein ähnliches aber automatisch extrahierbares Kohäsionsmodell bedient sich der Technik des *Slicings* [BiOt93], einer speziellen Datenflußanalyse: Die dort verwendete spezielle Form der *Daten-Slices* einer Variablen v ist gegeben durch die Sequenz aller *Daten-Token*, d.h. Variablen, Konstanten und Referenzen, die einen Effekt auf den Wert von v haben können (*Back Slice*) und aller Daten-Token, auf die der Wert von v einen Effekt haben kann (*Forward Slice*). Ein solcher Daten-Slice wird für jede Ausgabevariable (z.B. Rückgabewerte oder *Call-By-Reference*-Parameter) einer Funktion bestimmt. Das Verständnis von Kohäsion innerhalb dieses Produktmodells ist, daß diese Ausgabevariablen von möglichst vielen gemeinsamen Daten-Token abhängen. Tritt das Gegenteil ein, d.h. mehrere Ausgabeparameter haben keinen einzigen gemeinsamen Datentoken, so ist die Funktion bzgl. dieses Produktmodells überhaupt nicht kohäsiv, da die Ausgabeparameter völlig unabhängig voneinander bestimmt werden können. Würde diese nicht kohäsive Funktion für jede Ausgabevariable in Teilfunktionen zergliedert, entstünden kohäsivere Funktionen. Das diese Form von Kohäsion bestimmende Maß *SFC* (*strong functional cohesion*) ist entsprechend definiert als (*Anzahl der Daten-Token, die in allen Daten-Slices der Funktion vorkommen*) / (*Anzahl der Daten-Token*).

Für den Bereich der objektorientierten Systeme wird Kohäsion häufig auf der Klassenebene betrachtet, d.h. die Kohäsion von Methoden und Attributen innerhalb einer Klasse wird bestimmt. Die Produktmodelle der ersten Arbeiten dazu berücksichtigen die Attributbenutzungen von Methoden [ChKe94], entsprechen also dem Daten-Slicing, wobei auch hier nicht zwischen lesen (Forward Slice) und schreiben (Back Slice) unterschieden wird: Eine Klasse bzw. die in ihr implementierten Methoden werden als kohäsiv angesehen, wenn möglichst viele Methoden gemeinsame Attribute verwenden. Das weit verbreitete Maß *LCOM* (*lack of cohesion of methods,* [ebd.]) ist somit definiert als die Anzahl von Methodenpaaren, die kein Attribut gemeinsam verwenden, minus der Anzahl von Methodenpaaren, die wenigstens ein Attribut gemeinsam verwenden. Der Minimalwert von LCOM wird auf 0 normalisiert. Ein hoher LCOM-Wert entspricht folglich einem schlechten Kohäsionswert.

Aufgrund des empfohlenen Prinzips der Datenkapselung innerhalb von OO-Systemen, d.h. Attribute sollen nicht direkt, sondern über spezielle Zugriffsmethoden verwendet werden, mußte

das Kohäsionsmodell allerdings um die Benutzungsrelationen zwischen Methoden erweitert werden [HiMo95].
Noch spezifischer ist das Kohäsionsmodell von Bieman und Kang [BiKa95]: Hier wird jeweils für Methodenpaare, bei denen beide Methoden die Sichtbarkeit public besitzen, bestimmt, ob sie ein Attribut direkt oder indirekt gemeinsam benutzen, d.h. ob deren *cau* (*common attribute usage*) wahr, d.h. nicht die leere Menge ist. Das Maß *TCC* (*tight class cohesion*) ist darauf aufbauend definiert als das Verhältnis von cau-Methodenparen zu nicht-cau-Methodenpaaren. Je größer der TCC-Wert einer Klasse, desto kohäsiver ist sie.

3.3 Prozesse für einen erfolgreichen Produktmaßeinsatz

Die Bestimmung eines angepaßten Qualitätsmodells und die Selektion oder Erstellung geeigneter Maße geschieht nicht zum Selbstzweck, sondern soll den Entwickler bei der Erstellung und Wartung von Software unterstützen. Dieser Abschnitt soll das zu Beginn vorgestellte Prozeßmodell (s.o.) konkretisieren und mögliche Instanziierungen aufzeigen, bei denen Maße effektiv eingesetzt werden können. Da als Kriterium im Sinne des ökonomischen Prinzips (vgl. Kapitel 2) letztendlich die Produktqualität zählt und die Qualität der verwendeten Ressourcen und Prozesse ebenfalls häufig im Verhältnis zu dieser gesetzt wird, konzentriert sich diese Arbeit im weiteren auf Produktmaße.
Die Prozesse, bei denen Produktmaße effektiv einsetzbar sind und die größtenteils in eigenen Arbeiten angewendet wurden (vgl. auch Kapitel 9), können wie folgt klassifiziert werden:

- *Beurteilung von Produktqualität*, d.h. die Feststellung der graduellen Ausprägung von relevanten Qualitätsattributen entsprechend des angepaßten Qualitätsmodells zur Einschätzung der momentanen Produktqualität,
- *Vorbereiten eines Reviews*, d.h. die Identifikation von Produktteilen, deren Meßwerte entsprechend des angepaßten Qualitätsmodells als kritisch eingestuft werden und deren Betrachtung innerhalb eines durchzuführenden Reviews maximalen Nutzen verspricht,
- *Überarbeitung eines Programms*, d.h. die mit Meßwerten entsprechend des angepaßten Qualitätsmodells angereicherte Produktvisualisierung für ein zielorientiertes Verständnis des Programms und als Darstellungsform zur semiautomatischen Extraktion von Änderungsaktivitäten, und
- *Trendanalyse*, d.h. die Betrachtung des Produkts und dessen Meßwerte über verschiedene Versionen hinweg, um einen qualitativen Trend erkennen und gegebenenfalls Gegenmaßnahmen einleiten zu können.

Diese Prozesse werden im folgenden jeweils mit einigen konkreten Beispielen erläutert und detailliert beschrieben, um zu belegen, daß diese Klassifizierung von Prozessen eine wertvolle Strukturierung existierender Prozesse, bei denen Produktmaße eingesetzt werden, ermöglicht.

3.3.1 Beurteilung von Produktqualität

Innerhalb dieser Prozeßklasse wird das Produkt als gegebenes, fixes, d.h. nicht zu veränderndes Objekt betrachtet. Entsprechend eigener Qualitätsvorstellungen wird ein angepaßtes Qualitätsmodell erstellt, das sowohl relevante Maße als auch den jeweils gewünschten Wertebereich enthält. Das Vermessen des Produkts entsprechend dieser Vorgaben ermöglicht eine Einschätzung, inwieweit die Qualitätsanforderungen vom Produkt erfüllt werden. Diese Prozeßklasse verwendet i.d.R. Feedback Loops (vgl. Abschnitt 3), d.h. die Ergebnisse einer Messung werden analysiert und in weiteren Prozeßdurchläufen berücksichtigt.

Mögliche und in Projekten im Rahmen dieser Arbeit durchgeführte Szenarios einer Beurteilung von Produktqualität mittels Softwaremaßen sind:

- *Outsourcing-Beurteilung* (vgl. auch Kapitel 9.2.3): Kann innerhalb eines Softwareprojekts eine abgeschlossene Funktionalität identifiziert werden, für die fachlich zudem an anderer Stelle bereits genügend Erfahrung existiert, so kann eine Auslagerung dieses Teilprojekts sinnvoll sein. Diese Art von Softwareprojekten wird als *outsourced software* bezeichnet: „*The phrase outsourced software refers to software projects that are built under contract by an external company for a client organization* “ ([Jone00], S. 235). Diese Strategie ermöglicht ebenfalls, Ressourcen-Engpässen innerhalb des eigenen Unternehmens durch die projektspezifische Einbeziehung von Subunternehmen zu begegnen. Da das fertige Produkt allerdings vom Hauptunternehmer zusammengebaut, getestet und gewartet wird, d.h. der Hauptunternehmer direkt mit der internen Qualität des vom Subunternehmen produzierten Teilprodukts konfrontiert ist, ist die Einhaltung spezifischer Qualitätsanforderungen unumgänglich. Der Abnahmetest des vom Subunternehmer produzierten Teilprodukts umschließt damit häufig nicht nur den funktionalen Test, sondern ebenfalls die Überprüfung der im Vorfeld festgesetzten Qualitätsanforderungen. Diese werden bei einer derartigen Zusammenarbeit häufig bereits in die Produktanforderungen mit einbezogen.

  Eine konkrete Anwendung dieses Szenarios ist z.B. in [MaCo96] gegeben: Es wird ein vollständiger Prozeß beschrieben, mit dem der Telekommunikationsprovider *Bell Canada* mittels 22 Maßen die Qualität von durch Outsourcing erstellten Teilprodukten ermittelt und daraufhin über die Annahme oder Zurückweisung des Teilprodukts entscheidet. Das für diese Zwecke erstellte Werkzeug wird in Abschnitt 3.4.4 beschrieben.

  Dieses Szenario, in dem Softwaremaße eine herausragende Rolle spielen können, wird zunehmend wichtiger, da das Auslagern von Softwareteilprojekten immer häufiger angewendet wird. So berichtet z.B. die *Software-Productivity-Research-Company* folgende Zahlen: „[...] *approximately 15% of all information systems software in the United States is produced under contract or outsource agreements. This volume has increased from less than 5% circa 1989* “ ([Jone00], S. 241).
- *Meilenstein-Beurteilung*: Um innerhalb großer Projekte möglichst frühzeitig Abweichungen des Ist-Zustands vom Soll-Zustand festzustellen, damit gegebenenfalls Korrekturmöglichkeiten angewendet werden können (Feedback Loop), wird das Gesamtprojekt i.d.R. bereits während der Planung in mehrere, feingranularere Teilprojekte zergliedert. Das jeweils gewünschte Ergebnis eines dieser Teilprojekte wird als *Meilenstein* bezeichnet ([Somm92], S. 497f). Die dort formulierten Anforderungen können ebenfalls Angaben zu den Qualitätsanforderungen enthalten. So kann z.B. der Meilenstein, der nach der Entwurfsphase erreicht sein soll, neben der Vollständigkeit der berücksichtigten Funktionen und Daten auch projektspezifische Qualitätsmerkmale für Entwürfe beinhalten. Werden diese bei der Überprüfung des Meilensteins nicht eingehalten, so gilt er als nicht erreicht und das Produkt bedarf einer Überarbeitung. Der Vorteil dieser frühen Erkennung von Qualitätsverletzungen liegt in der kostengünstigen Behebung. Die Alternative der Behebung erst während der Test- oder sogar Wartungsphase ist um ein vielfaches teurer.

Positive Erfahrungen mit diesem Szenario sind z.B. in [Dask92] beschrieben: Dort werden für einige Projekte des Unternehmens *Motorola* die groben Meilensteine wie z.B. Anforderungsanalyse oder Entwurf erläutert und entsprechende Qualitätsanforderungen beschrieben (z.B. "*Requirements tracking metrics* " und "*Design Tracking metrics* ", [ebd.]). Die damit innerhalb von Motorola verbesserten Möglichkeiten des Projektmanagements haben für einzelne Produkte eine Fehlerreduktion in den Auslieferungen um den Faktor 50 gebracht, d.h. die Anzahl im Systemeinsatz aufgetretener Fehler ist auf zwei Prozent des ursprünglichen Wertes zurückgegangen.

- *Standard-Einhaltung* (vgl. Kapitel 9.2.1): Für bestimmte Klassen von Anwendungen beginnen sich Standards zu etablieren, die grobe Vorgaben für die einzuhaltende Qualität machen. Auch wenn diese zur Zeit noch größtenteils nur die Ebene der Kodierungsrichtlinien (z.B. Namenskonventionen, Einrückungsstile und Dokumentationsschablonen) abdecken, werden längerfristig ebenfalls meßbasierte Qualitätsanforderungen standardisiert werden. Die ISO 9126 ist ein erster, wenn auch aufgrund der fehlenden konkreten Maße noch unvollständiger Schritt. Diese Unvollständigkeit soll aber mit zukünftiger Erfahrung behoben werden können: "*Der Stand der Technik gestattet es noch nicht, quantitative Meßgrößen für alle Merkmale zu normen.* [...] *Die angegebenen Teilmerkmale werden zur Anwendung empfohlen, um Erfahrungen für künftige Verfeinerungen der Norm zu sammeln* " ([ISO9126], Vorwort).
  Aus Unternehmer-Sicht ist ein wesentlicher Vorteil einer solchen Standardeinhaltung die verbesserte Marktposition: So wie z.B. das amerikanische Verteidigungsministerium für Softwareprojekte nur Firmen beauftragen darf, deren Prozesse den CMM-Level 3 erreichen ([Jone00], S. 480), so könnte ebenfalls für die Produkte ein solcher Standard eingeführt werden, der dann als notwendiges Kriterium bei der Produktauswahl dient.
  Wichtige Arbeiten in diese Richtung wurden innerhalb des Esprit-Projekts *SCOPE* (*Software assessment and Certification Programme in Europe*) gemacht [RaHaRo95]. Schwerpunkt war die Feststellung der Konformität eines Produkts zur ISO 9126. Neben konzeptuellen Arbeiten an Methoden, wie die Qualität von Produkten beurteilt werden kann, und konkreten Maßkatalogen, die angewendet wurden, sind die Projektergebnisse in über 20 groß angelegten Feldversuchen empirisch getestet worden. Das Ergebnis des SCOPE-Projekts war ein 5-Schritt-Evaluationsvorgehen, das größtenteils in die ISO 14598 übernommen wurde. Es umfaßt die Analyse der Evaluationsanforderungen, die Spezifikation und den Entwurf des Evaluationsprozesses sowie die Anfertigung der dabei zu erarbeitenden Dokumente. Das Vorgehensschema enthält Konzepte zur Bestimmung der *Evaluationstiefe* (s.u.) und zur Modularisierung des Evaluationsprozesses (*Evaluationsmodule*). Diese Evaluationsmodule sind die Basis für eine mögliche Zertifizierung. Sie kapseln folgende Informationen:
  - Beschreibung des gemessenen Qualitätsmerkmals, des verwendeten Maßes, dessen Akzeptanzkriterien und der für das Vermessen notwendigen Technik,
  - Beschreibung der Evaluationstiefe, d.h. die Art der Auswirkungen einer Fehlfunktion, die von vernachlässigbar (*Level D*) bis zu lebensgefährlich (*Level A*) reicht,
  - Beschreibung der Anforderungen an ein Produkt, für das das jeweilige Evaluationsmodul angewendet werden können soll,
  - Beschreibung der Meßdokumentation für das Maß und
  - Beschreibung des Aufwands zur Bestimmung der Meßwerte.

  Nach der Selektion der für ein zu untersuchendes Projekt angepaßten Evaluationsmodule werden diese wie folgt angewendet:

- Das Produkt wird mittels der in den ausgewählten Evaluationsmodulen enthaltenen Maße vermessen,
- die Ergebniswerte werden mit den für die Maße jeweils angegebenen Akzeptanzkriterien verglichen und
- eine der Vorgabe entsprechende Meßdokumentation wird erstellt.

Statt eines Standards existiert entsprechend der unterschiedlichen Arten von Softwareprodukten eine Vielzahl von verfeinerten Standards, die gegeben sind durch unterschiedliche Mengen von Evaluationsmodulen.

3.3.2 Vorbereitung eines Reviews

Während das Vermessen von Softwareprodukten häufig vollständig automatisiert durchgeführt werden kann, ist eine andere effektive Maßnahme zur Qualitätssicherung von Softwareprodukten durch das bewußt manuell durchzuführende *Review* gegeben:

> *„A review is a process or meeting during which a work product, or a set of work products, is presented to project personnel, managers, users, customers, or other interested parties for comment or approval.“*

Definition 10: Review (aus IEEE Standard Glossary 1990, in [Marc94], S. 1084)

Spezielle Formen des Reviews sind

- der *Walkthrough*, d.h. eine statische Analysetechnik, bei dem der Designer oder Programmierer einige Projektmitglieder durch eine Dokumentation oder einen Quelltext führt, wobei die Zuhörer Fragen stellen und Kommentare über mögliche Fehler, Standardverletzungen oder andere Probleme abgeben (IEEE Standard Glossary 1990, in [Marc94], S. 1084),
- die *Inspection*, d.h. eine statische Analysetechnik, bei der das zu inspizierende Produkt betrachtet wird, um Fehler, Standardverletzungen oder andere Fehler zu finden (IEEE Standard Glossary 1990, in [Marc94], S. 1084) und
- das *Audit*, eine unabhängige Untersuchung eines Produkts, um dessen Konformität zu Spezifikationen, Standards, Vertragsteilen oder anderen Kriterien zu belegen und Schwachstellen aufzuzeigen (IEEE Standard Glossary 1990, in [Marc94], S. 1084). Während die Inspection häufig und eher informell durchgeführt wird, werden Audits in formellerem Rahmen und zu besonderen Entscheidungspunkten durchgeführt (z.B. Abnahmetest oder Outsourcing-Beurteilung).

Alle drei Formen des Reviews sind zwar aufgrund der intensiven Betrachtung sehr gut geeignet, um qualitative Schwachstellen zu identifizieren, allerdings ist ein komplettes Review für große Systeme häufig nicht praktikabel: So wird z.B. in [FrLuSa95] der Aufwand eines kompletten Reviews mit bis zu 20% des gesamten Erstellungsaufwands angegeben.
Softwaremaße können hier eine sehr interessante Aufgabe übernehmen: Die Grundannahme dabei ist, daß ein Review für unterschiedliche Programmfragmente unterschiedlich effektiv ist, d.h., daß z.B. ein Review von sehr einfachen und wenig kritischen Teilen sehr viel weniger Schwachstellen identifiziert als ein Review für einen sehr komplexen und relevanten Programmteil. Da die Ressourcen für ein Review i.d.R. nicht in beliebigem Umfang zur Verfügung stehen, ist ein Review mit fixem Ressourcenumfang dann am effektivsten, wenn es sich auf die komplexesten und relevantesten Teile konzentriert. Die Selektion dieser Programmteile für das Review sollte seinerseits möglichst wenig Aufwand kosten. Eine Möglichkeit, diese Selektion semiautomatisch mittels Softwaremaßen vorzunehmen ist in eigenen Arbeiten vorgestellt und durchgeführt worden [SiRuKö98]. Der dort angewendete Prozeß für ein meßbasiertes Audit läßt sich als Datenflußmodell vereinfacht wie folgt darstellen:

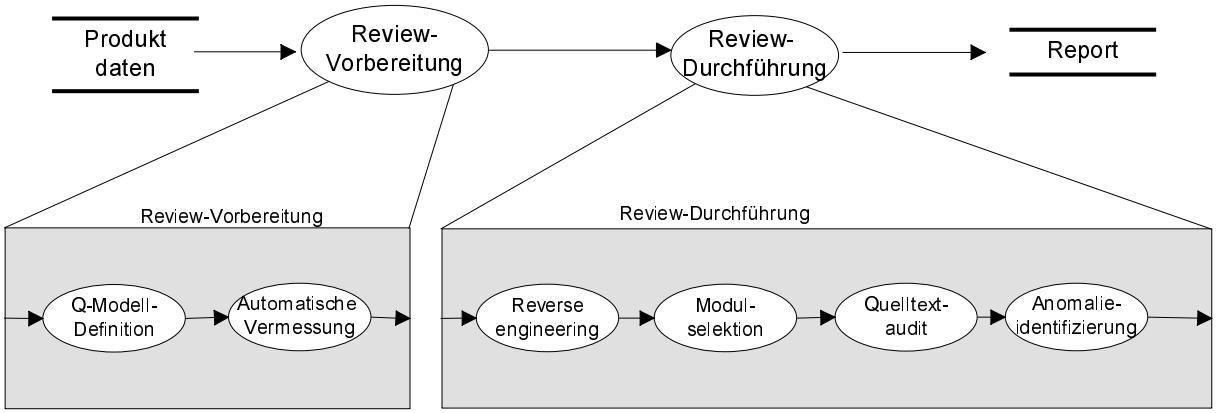


Abbildung 7: Datenflußmodell einer meßbasierten Review-Vorbereitung

Die Schritte bedeuten im einzelnen:

1 *Review-Vorbereitung*: Die Produktdaten liegen als Quelltext des zu begutachtenden Programms vor. Die Review-Vorbereitung erweitert diese Informationen durch folgende Teilschritte:
   1.1 *Q-Modell-Definition*: Hier wird explizit festgelegt, welche qualitativen Anforderungen an das zu begutachtende Produkt gestellt werden. Das Ergebnis ist ein angepaßtes Qualitätsmodell (vgl. Kapitel 2), an dessen Blättern Produktmaße und die gewünschten Wertebereiche notiert sind.
   1.2 *Automatische Vermessung*: Das angepaßte Qualitätsmodell dient als Konfiguration eines Meßwerkzeugs, das die Produktdaten entsprechend vermißt und die Meßwerte derart darstellt, daß eine effiziente Interpretation und Konzentration auf die für das Review relevanten Teile gefördert wird (vgl. Abschnitt 3.4).

2 *Review-Durchführung*: Die eigentliche Review-Durchführung erhält als Eingabe die Produktdaten (in Form des Quelltextes), das angepaßte Qualitätsmodell und die aufbereiteten Meßergebnisse. Das Ergebnis der Review-Durchführung ist der Report, in dem die gefundenen Qualitätsverletzungen dokumentiert sind.
   Die Review-Durchführung teilt sich in folgende Teilschritte auf:
   2.1 *Reverse Engineering*: Um einen groben Überblick über das System insgesamt zu erhalten, werden in diesem Schritt Module in Form von Subsystemen identifiziert, deren Zusammenspiel und Abhängigkeiten untersucht, diese eventuell mit zusätzlichem Anwendungsfallwissen angereichert und so das gesamte System auf einem höheren Abstraktionsniveau dargestellt. Dieses Reverse-Engineering kann ebenfalls meßunterstützt geschehen (vgl. Abschnitt 3.3.3).
   2.2 *Modul-Selektion*: Es werden diejenigen Module ausgewählt, die entsprechend der Meßwerte und damit entsprechend der qualitativen Anforderungen an das Produkt besonders negativ auffällig sind. Grundlage der Entscheidung sind die aufbereiteten Meßdaten.
   2.3 *Quelltext-Audit*: Der anschließende klassische und zeitintensive Quelltext-Audit findet ausschließlich für die auffälligen Softwaremodule statt und orientiert sich an den Qualitätskriterien des festgelegten Qualitätsmodells.
   2.4 *Anomalie-Identifizierung*: Hier werden die im Audit gefundenen Abweichungen vom Qualitätsmodell notiert und erläutert. Diese sind das Hauptergebnis des Reviews und beinhalten eine Liste von Schwachstellen, deren Behebung die Softwarequalität erhöht.

In [SiRuKö98] wird der Vorteil dieser meßbasierten Review-Vorbereitung anhand eines Versuchs praktisch belegt: So findet bei jeweils gleichem Aufwand ein Review eines derartig vorbereiteten

Produkts deutlich mehr Qualitätsverletzungen als ein Review von zufällig ausgewählten Softwareteilen. Somit ist es möglich, mittels Softwaremaßen die Effektivität von Reviews deutlich zu verbessern.

3.3.3 Überarbeitung eines Programms

Ein Schlüsselelement bei der Berücksichtigung von Qualität innerhalb von Softwareprodukten sind abgeleitete, konstruktive Maßnahmen, die mögliche Softwaremodifikationen betreffen und deren Durchführung positiven Einfluß auf die Qualität hat[4]. Neben der Berücksichtigung solcher qualitätserhöhenden Maßnahmen während der Softwareerstellung spielt die nachträgliche Veränderung eines Softwareprodukts eine wesentliche praktische Rolle (vgl. Kapitel 2). Die Anwendung solcher Maßnahmen an existierenden Systemen wird als eine Teildisziplin des Software-Engineering mit dem Begriff *Re-Engineering* bezeichnet:

> *„Re-Engineering ist die Untersuchung und Änderung eines bestehenden Systems mit dem Ziel, das System in einer neuen, geänderten Form zu implementieren. Dies kann sich auch auf die Adaptierung des Systems an geänderte Anforderungen erstrecken.*"

Definition 11: Re-Engineering, aus ([KlGa95], Glossar)

Aufgrund des iterativen Charakters von objektorientierter Softwareentwicklung spielt das Re-Engineering hier auch während der Erstellung selbst eine inhärent wichtige Aufgabe. Neuere Prozeßmodelle wie z.B. das *Fontänenmodell* [HeEd93] oder das *Extreme Programming* [Beck00] fördern ebenfalls das permanente Überarbeiten von Produkten anstelle der auf Anhieb zu erzielenden Produkt-Perfektionalität (vgl. [LuNe00]). Im Gegensatz zur Review-Vorbereitung (vgl. Abschnitt 3.3.2), bei dem die Selektion kritischer Teile für einen Report im Vordergrund steht, werden hier konkrete Re-Engineering-Tätigkeiten abgeleitet, selbst durchgeführt und bzgl. der durchgeführten Veränderung untersucht. Daher ist es möglich, auch bereits sehr präzise, eventuell aber auch sehr zahlreiche Maßnahmen vorzuschlagen und durchzuführen. Diese Mini-Re-Engineering-Maßnahmen, die jeweils durch Maße motiviert sind und innerhalb einiger Minuten durchgeführt werden können, werden als *Refactorings* bezeichnet [Fowl99] und werden in zunehmendem Maße wichtig für die Softwareentwicklung großer Systeme (vgl. z.B. [ebd.] und [LuNe00]). Softwaremaße können auch hier die Schlüsselrolle spielen, da sie den Entscheidungsprozeß unterstützen können, an welcher Stelle welche *Refactorings* wann durchgeführt werden müssen (vgl. dazu auch eigene Erfahrungen in Kapitel 9.3).

Bezüglich des zeitlichen Aufwands einer Re-Engineering-Maßnahme spielt – neben der Durchführung der Änderungsmaßnahme selbst – besonders das Verständnis des Quelltextes, an dem die Änderung vorgenommen werden soll, eine große Rolle. In [Fraz92] wird z.B. die in Abbildung 8 dargestellte Aufwands-verteilung für die verschiedenen

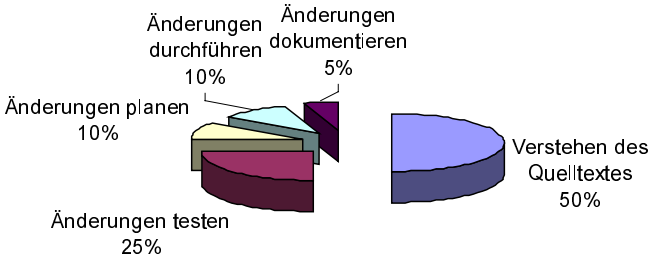


Abbildung 8: Aufwandsverteilung beim Re-Engineering

[4] Mit Maßen ist ebenfalls eine Unterstützung der grundsätzlichen Entscheidung über eine mögliche Weiterarbeit an einem Programm möglich (ob z.B. ein Programm überhaupt überarbeitet oder doch ausgemustert werden soll). Die von Jacobson und Lindström vorgestellte *Entscheidungsmatrix* besitzt z.B. die Entscheidungsfaktoren *Anwendungswert* und *Veränderbarkeit*; letzterer ist, nach Verfeinerung mit einem Qualitätsmodell, meßbasiert bestimmbar [JaLi91]. Dasselbe gilt für die von Schmidt und Postema vorgestellte *erweiterte Entscheidungsmatrix* und deren zusätzlichen Faktor der *Kapselung* (für die Bestimmung des Wiederverwendungswertes) [ScPo98].

während des Re-Engineering anfallenden Tätigkeiten berichtet: Der Aufwand beim Re-Engineering liegt dementsprechend zu 50% im bloßen Verstehen des Quelltextes. Softwaremaße können hier helfen, diesen Aufwand zu reduzieren bzw. bei gleichem Aufwand das Verständnis zu vergrößern. Dies gilt speziell für die Gewinnung einer Übersicht, die es ermöglicht, das Gesamtsystem als Komposition kleinerer und besser handhabbarer Teilsysteme aufzufassen, deren Zusammenspiel zu verstehen und diese Kenntnis in entsprechende Darstellungen abzulegen. Diese Aktivität geht als frühe Teilaktivität des Re-Engineering jeder durchzuführenden Änderung voraus und wird als *Reverse-Engineering* bezeichnet:

> „*Unter Reverse-Engineering versteht man den Prozeß der Analyse eines bestehenden Systems, mit dem Zweck der Identifikation von Systemkomponenten und deren Beziehungen untereinander, sowie der Erzeugung von Darstellungen des untersuchten Systems auf unterschiedlichen, höheren Abstraktionsniveaus.*"

Definition 12: Reverse-Engineering, aus ([KlGa95], Glossar)

Für ein erfolgreiches Reverse-Engineering, d.h. die möglichst präzise Analyse des Systems mit möglichst minimalem Aufwand, ist besonders die statische, automatische *Softwarevisualisierung* interessant, da der Entwickler an ihr sehr viel einfacher Strukturen und Zusammenhänge erkennen kann:

> Als *Softwarevisualisierung* wird die Transformation von vorhandenen, auf Software bezogene Informationen in eine sinnvolle und gebrauchstaugliche visuelle (nicht zwangsläufig bildliche) Darstellung verstanden, die es dem Benutzer erlaubt, die Ausgangsinformationen zu verstehen.

Definition 13: Softwarevisualisierung (in Anlehnung an [SDBP98])

Softwarevisualisierung meint also weniger das Benutzen von Bildern als Medium als die Möglichkeit, aus dem Medium ein mentales Bild zu kreieren. Allgemeiner können an Visualisierungen die Anforderungen gestellt werden, daß sie

- ein Vokabular definierter (graphischer) Elemente (Alphabet) benötigen und
- Regeln besitzen, nach denen die Elemente kombiniert werden (Grammatik) (vgl. [ebd.]).

Die Menge der für eine solche automatisch generierte statische Softwarevisualisierung genutzten Daten, die für das Alphabet und die Grammatik verwendet werden, kann in folgende Typen unterteilt werden:

- Hierarchische Systemstrukturdaten, die das Gesamtsystem mittels einer hierarchischen Enthaltenseinstruktur über verschiedene Abstraktionsstufen verfeinern,
- unterschiedliche Abhängigkeiten zwischen Teilen des Systems (z.B. durch Benutzung) und
- Anreicherung der dargestellten Teile um aufgabenspezifische und abstraktionsstufenabhängige Meßwerte.

In [SiRuKö99a] werden z.B. folgende Daten für eine automatisch generierte Visualisierung objektorientierter Systeme verwendet:

- Als Systemhierarchie wird folgende Enthaltenseinkette dargestellt: System $\supseteq$ Subsystem $\supseteq$ Datei $\supseteq$ Klasse $\supseteq$ Methode/Attribut.
- Als Abhängigkeiten zwischen Teilen des Systems werden Benutztbeziehungen verwendet, speziell für die Klassenebene zusätzlich die Vererbungsbeziehungen.
- Die betrachteten Teile werden jeweils mit Größen- und Kopplungsmaßen angereichert.

In einem dort beschriebenen Re-Engineering-Versuch, vorhandene Klassen eines bestehenden, objektorientierten Systems bzgl. ihrer Funktionalität in Subsysteme aufzuteilen, waren die ausschließlich auf dieser Visualisierung basierenden Restrukturierungen vergleichbar mit denen,

die sich durch direkte Quelltextbetrachtung ergeben haben. Zusätzlich aber war die Verwendung der Visualisierung effizienter und ermöglichte die Berücksichtigung mehrerer Aspekte, die für eine Subsystembildung relevant sind. Die Einbindung von Meßwerten in die Visualisierung wurde von den Versuchsteilnehmern als ein Schlüsselelement für ein erfolgreiches Reverse-Engineering identifiziert (vgl. [ebd.]).

Eine ähnliche Unterstützung der Umstrukturierung von Software durch Softwaremaße innerhalb einer Softwarevisualisierung wird in [Finn et al.97] beschrieben. Der dort vorgestellte Visualisierungsansatz reichert allerdings die für die Grundstruktur verwendeten und automatisch extrahierbaren Daten mit semantischen Informationen aus Interviews, Sitzungsprotokollen oder externen Dokumentationen an. Deren Gewinnung ist jedoch sehr aufwendig.
Die für die Grundstruktur verwendeten Daten lassen sich allerdings wiederum entsprechend obiger Klassifizierung beschreiben:

- Als Systemhierarchie wird die Enthaltenseinkette Domäne $\supseteq$ Entwurf $\supseteq$ Programm $\supseteq$ Rohdaten verwendet.
- Für die Darstellung von Abhängigkeiten werden existierende Produkte wie z.B. *Rigi* verwendet (vgl. [MüKl88]), so daß diese über die Art der betrachteten Abhängigkeiten entscheiden. Rigi selbst verwendet Graphen als allgemeine Darstellung der Strukturen und erlaubt verschiedene Aspekte für die Kantendefinition (z.B. Benutztbeziehungen und Include-Beziehungen).
- Die betrachteten Teile werden jeweils mit Größenmaßen (z.B. McCabe) und Kopplungsmaßen (z.B. fan-in und fan-out) angereichert.

Softwaremaße bieten aufgrund der häufig numerischen Meßwerte insgesamt eine ausgezeichnete Grundlage, das Reverse-Engineering und damit das Re-Engineering zu unterstützen. Neue, aussagekräftigere und integrierte Darstellungen von Struktur und Meßwerten werden in Kapitel 5.3 und 9.1 detailliert vorgestellt. Deren konkrete Einbeziehung in spezielle Restrukturierungsmaßnahmen ist Thema des Kapitels 9.3.

3.3.4 Trendanalyse

Sowohl durch Softwarewartung nach der Produktauslieferung als auch allgemein durch Re-Engineering-Maßnahmen während der gesamten Softwarebearbeitung entstehen verschiedene *Versionen* des zugrundeliegenden Produkts, d.h. zeitlich nacheinander liegende Ausprägungen eines Software-Elements (vgl. [Balz98], S. 238). Wird das Produkt innerhalb jeder Version vermessen und die Meßwerte archiviert, so ergibt sich daraus eine Sequenz von Meßwerten, der als *Trend* bezeichnet wird (vgl. [ErLe96]).
Die Analyse dieser Meßtrends für ein Maß einer Entität basiert i.d.R. auf drei wiederkehrenden Mustern:

- Die Meßwerte verändern sich nicht, d.h. die durch das Maß gemessene Eigenschaft ist über die betrachteten Versionen hinweg invariant geblieben. Häufig werden innerhalb der Trendanalyse solche Trends ausgeblendet, da dort meistens die Begutachtung von qualitativen Änderungen betrachtet werden soll, bei denen sich – bei passender Maßauswahl - folglich auch die Meßwerte verändern.
- Die Meßwerte verbessern sich, d.h. die durch das Maß gemessene Eigenschaft ist über die betrachteten Versionen hinweg verbessert worden. Mit diesem Analysemuster werden i.d.R. Re-Engineering-Phasen bestätigt, die für die qualitative Verbesserung des bestehenden Systems durchgeführt wurden. Für eine Trendanalyse mit dem Zweck der frühen Gefahrenerkennung können diese Trends allerdings auch ausgeblendet werden.
- Die Meßwerte verschlechtern sich, d.h. die durch das Maß gemessene Eigenschaft ist über die betrachteten Versionen hinweg schlechter geworden. Dieses Muster ist das interessanteste, da es einen früheren Einblick in mögliche Qualitätsdefizite erlaubt, als wenn erst mögliche

Meßwertgrenzwerte überschritten sind, was wiederum eine kostengünstigere Behebung des Defizits verspricht.

Durch die Konzentration auf eines dieser Muster erlaubt die Trendanalyse die starke Reduzierung der durch Messen entstehenden Datenmenge. So wird z.B. in [ErLe96] von einem Experiment berichtet, bei dem durch eine Trendanalyse und einer Ausblendung aller nicht veränderten Werte die Anzahl der Meßergebnisse drastisch reduziert werden konnte.
Neben der Datenreduktion durch eine Trendanalyse ist die Visualisierung geeignet, die gleichartigen Daten, die über mehrere Versionen ermittelt werden, aussagekräftig darzustellen: Neben zweidimensionalen *Trend-Charts*, die die Werte der verschiedenen Versionen als Kurven darstellen und deren Steigung Aussagen über den Trend erlaubt [SiRuLe00], ermöglicht diese Art von Information neue Visualisierungsformen wie z.B. dreidimensionale Welten, in denen die dritte Dimension den zeitlichen Verlauf darstellt, oder farblich angereicherte Bilder, in denen die Farbe mit der Veränderungshäufigkeit der betrachteten Komponente korreliert [JaGaRi99].
Das Hauptproblem der Trend-Analyse ist die automatische Verfolgung der Komponenten über verschiedene Versionen, da diese durch Umbenennung, Verschiebung oder auch Zusammenlegung mit anderen Komponenten nicht immer durch einfache Namensgleichheit identifiziert werden können (vgl. [SiRuLe00]). Neuere, noch im Versuchsstadium befindliche Ansätze zeigen diesbezüglich zwar ein deutliches Potential der Automatisierung, benötigen aber immer noch die manuelle Nachbearbeitung der ermittelten Komponentenverfolgungen [AnCaLu98].

3.4 Meßwerkzeuge

Ein wesentlicher Vorteil der Softwarevermessung ist die Möglichkeit der Automatisierung, da so selbst größte Datenvolumen mit vertretbarem Aufwand betrachtet werden können, um dann entsprechende Interpretationen zuzulassen, die ihrerseits entsprechende Aktionen begründen können (vgl. Feedback Loop, Abschnitt 3). Der Schlüssel für diese Automatisierung liegt in der Verwendung geeigneter Werkzeuge: Ganz allgemein lassen sich die Werkzeuge, mit denen Softwarevermessung möglich ist, wie folgt definieren:

> „*Ein Softwaremeßtool* [synonym zu *Softwaremeßwerkzeug*, Anm.d.A.] *sei ein Softwarewerkzeug, welches Komponenten eines Softwareprodukts oder der Softwareentwicklung in ihrer Quellform oder transformierten Form (z.B. als spezielles Modell) einliest und nach vorgegebenen Verarbeitungsvorschriften numerisch oder symbolisch auswertet.*"

Definition 14: Definition von Softwaremeßwerkzeug, aus [DFKW96], S. 39

So wie sich die Softwaremaße, d.h. die Art und Weise, mit der einem Beobachtungsobjekt ein Meßwert zugeordnet wird, in die drei Klassen Prozeß-, Ressourcen- und Produktmaße einteilen lassen (vgl. Abschnitt 3.1), so können auch die sie jeweils implementierenden Softwaremeßwerkzeuge klassifiziert werden in

- *Softwareprozeß-Meßwerkzeuge* für die Aktivitäten während der Softwarebearbeitung,
- *Softwareressourcen-Meßwerkzeuge* für die Mittel der einzelnen Prozesse und
- *Softwareprodukt-Meßwerkzeuge* für die Ergebnisse von Prozessen.

So wie diese Grenzen der Klassifizierung für Softwaremaße nicht fix sind (z.B. bei indirekten Maßen, die sich aus einer Kombination von Maßen aus evtl. unterschiedlichen Klassen errechnen lassen), so kann diese Klassifizierung für die Softwaremeßwerkzeuge ebenfalls nur als grobe Richtlinie dienen. Dies um so mehr, da die Zuordnung eines Softwaremeßwerkzeugs zu einer der drei Klassen häufig von der konkreten Art der Anwendung abhängt: So läßt sich z.B. ein

Werkzeug zur Quelltextvermessung ebenfalls als Ressourcenvermessung verwenden, indem durch das Produkt indirekt die Produktivität der Entwickler abgeschätzt wird.

Eine wesentliche Funktionalität der Softwaremeßwerkzeuge ist durch den eigentlichen Zweck des Vermessens motiviert, eine vereinfachte Übersicht über die graduellen Ausprägungen bestimmter Eigenschaften zu erhalten: Die aussagekräftige und anpaßbare Visualisierung (vgl. oben) der ermittelten Meßwerte. Deren Notwendigkeit steigt durch die von Softwaremeßwerkzeugen gegebene Möglichkeit, sehr große Mengen von Entitäten zu vermessen, da der Vorteil von Softwaremaßen, d.h. die starke Abstraktion und Datenreduktion der Ausgangsentitäten, durch eine übergroße Menge von Meßwerten zunichte gemacht werden kann. Diese für die Anwendung von Maßen – und hier besonders von Produktmaßen – in der Praxis große Gefahr wird innerhalb der verschiedenen Softwaremeßwerkzeuge deutlich unterschätzt. In eigenen praktischen Projekten hat sich jeweils sehr frühzeitig die Notwendigkeit der zielgerichteten Aufbereitung der Meßdaten gezeigt. Diese, entsprechend des in Definition 13 erläuterten Visualisierungsbegriffs als notwendige Transformation von vorhandenen Informationen in eine sinnvolle und gebrauchstaugliche visuelle Darstellung bezeichnet, kann durch folgende, beliebig kombinierbare und in den eigenen entwickelten Werkzeugen implementierten Techniken unterstützt werden [SiRuLe00]:

- *Statistik*: Da die Meßwerte häufig in numerischer Form vorliegen, können viele mathematische Verfahren zur Analyse der Daten angewandt werden. Dies umfaßt Funktionen wie z.B. arithmetisches Mittel, Standardabweichung oder Häufigkeitsanalysen. Für die Berechnung der letzteren genügen sogar nicht-numerische Meßwerte (vgl. Nominalskala in Kapitel 4).
  Solche Techniken dienen primär der qualitativen Übersicht über das Gesamtsystem, da einzelne Entitäten nicht direkt identifiziert werden können; sie werden daher intensiv für die Prozesse der Produktqualitätsbewertung verwendet (vgl. Abschnitt 3.3.1).
- *Wertefilterung*: Häufig sind für spezielle Handlungsziele nicht alle ermittelten Meßwerte interessant, sondern nur besondere Wertintervalle. Diese können z.B. innerhalb der angepaßten Qualitätsmodelle an den zu verwendenden Maßen bereits in Form der gewünschten Wertebereiche notiert sein (vgl. Kapitel 2). Die Filter können dabei absolut oder relativ angegeben werden.
  Solche Techniken dienen primär der Fokussierung auf einzelne Meßwerte und den dahinter stehenden Entitäten und werden daher intensiv für die Review-Vorbereitung verwendet (vgl. Abschnitt 3.3.2).
- *Komponentenfilterung*: Ein Softwaresystem ist i.d.R. aus vielen Teilsystemen zusammengesetzt, deren Vermessung unterschiedlich interessant sein kann: So können z.B. die Meßwerte eines eingebauten, fixen Produkts (z.B. eine Bibliothek oder ein Framework) uninteressant sein, da ein solches System häufig nicht verändert werden kann oder darf. Daraus folgt, daß häufig bereits im Vorfeld einer Messung klar ist, für welche Systemteile bzgl. einer Maßnahme die Meßwerte überhaupt interessant sind. Die Analyse wird demzufolge aussagekräftiger und effektiver, wenn nur diejenigen Meßwerte untersucht werden, die für den jeweiligen Kontext interessant sind. Die Bedeutung dieser Technik wird besonders in den Fallstudien in Kapitel 9 deutlich.
- *Diagramme*: Die Interpretation numerischer Werte kann durch diagrammbasierte Darstellungen vereinfacht werden. Typische Diagrammarten sind (vgl. [SiRuLe00], [Eber92]):
  - *Symboldiagramme* für eine Zuordnung von Meßobjekt zu Meßbereich,
  - *Balkendiagramme* für eine Zuordnung von Meßobjekt zu Meßwert,
  - *Verteilungsdiagramme* für eine Zuordnung von Meßwertintervall zu Auftrittshäufigkeit,
  - *Kiviatdiagramme* für eine Zuordnung von einem Meßobjekt zu vielen Meßwerten,

- *Kurvendiagramme* für eine Zuordnung von einem Meßobjekt zu vielen Meßwerten in einem vergleichbaren Wertebereich und
- *Scatterplots* für eine Analyse von mehreren Meßwerten mehrerer Maße eines Meßobjekts.

In den folgenden Abschnitten sollen exemplarisch Werkzeuge für die jeweilig vermessenen Entitäten vorgestellt werden: Abschnitt 3.4.1 zeigt ein Beispiel eines Softwaremeßwerkzeugs für die Aktivitäten während der Softwarebearbeitung, Abschnitt 3.4.2 stellt ein Beispiel eines Softwareressourcen-Meßwerkzeugs vor und Abschnitt 3.4.3 behandelt vertieft den Bereich der Softwareprodukt-Meßwerkzeuge (inkl. der Vorstellung von acht, für die Beschreibung von Software-Produktmeßwerkzeugen relevanten Funktionsaspekten), der in Abschnitt 3.5 durch ein selbst entwickeltes Werkzeug ergänzt und mit eigenen Erfahrungen angereichert wird.

3.4.1 Beispiel eines Softwareprozeß-Meßwerkzeuges

Ein bekanntes Werkzeug – das *ami-Tool* – zur Vermessung von Softwareprozessen ist das Ergebnis des ESPRIT-Projekts *ami* (*application of metrics in industry*), das die meßbasierte Unterstützung des Softwaremanagements als Ziel hatte [PKSH95]. Es faßt die Erfahrungen aus verschiedenen europäischen Projekten wie z.B. *METKIT, MUSE, PYRAMID* und *MUSiC* zusammen. Die Funktionalität des ami-tools kann dabei in die folgenden, sequentiell durchzuführenden Teilfunktionalitäten gegliedert werden:

1. Beurteilung der Güte des aktuell verwendeten Prozesses im Rahmen des CMM-Modells: Die Beurteilung geschieht dabei entlang der sieben Dimensionen *Prozeßorganisation*, *Prozeßsteuerung*, *Datenmanagement*, *Prozeßquantifizierung*, *Standardisierungsniveau*, *Technologiebeherrschung* und *Ressourcenniveau*, deren Teilgüte jeweils mittels spezieller, anpaßbarer Fragenkataloge ermittelt wird. Die Fragen werden dabei jeweils manuell in speziellen Eingabemasken beantwortet.
2. Analyse der Zielsetzung bzgl. der Verbesserung des aktuell verwendeten Prozesses mittels der GQM-Methode (vgl. Kapitel 2.3): Feststellung der beabsichtigten Ziele, der zu den Zielen führenden Fragen und schließlich der die Fragen zu beantworten helfenden Maße. Innerhalb des ami-Tools wird diese Struktur (noch ohne die Maße) als Baum vorgegeben, wobei an den Blättern möglichst feingranulare Merkmale liegen, d.h. konkrete Prozeß-, Produkt- und Ressourcenmerkmale.
3. Auf Grundlage der CMM-Bewertung und des Zielbaums können nun geeignete Maße für die Bestimmung der feingranularen Merkmale an den Blättern identifiziert werden. Darauf aufbauend kann ein Meßplan erstellt werden, der als Anleitung für die Durchführung und Auswertung der jeweiligen Messungen dient.
4. Verbesserung entsprechend der gesetzten Ziele. Deren Erfolg kann durch eine erneute Prozeßbewertung bestätigt werden.

Da ein Großteil der Prozeßattribute grundsätzlich nicht automatisch extrahierbar sind, basieren Softwareprozeß-Meßwerkzeuge im allgemeinen auf manueller Dateneingabe. Im konkreten Fall von ami-tool bedeutet das die Vermessung durch formulargestützte Eingaben vorhandener Daten (z.B. Punkt 1 und 3). Ein Großteil der Funktionalität innerhalb dieser Klasse von Werkzeugen liegt daher in der computergestützten Dateneingabe, deren Vermessung, der persistenten Haltung für historische Analysen und der zielgerichteten Visualisierung.

3.4.2 Beispiel eines Softwareressourcen-Meßwerkzeuges

Wie bereits das ami-Tool basiert das Werkzeug *Micro Planning* auf der projektbegleitenden Datenerfassung und Analyse (vgl. [Mosi96]). Neben prozeßspezifischen Analysen setzt Micro Planning allerdings den Schwerpunkt auf verschiedene Techniken der optimalen Ressourcenzuweisung, der automatischen Verteilung von Teilfunktionalitäten innerhalb eines Zeitraums, worst-case Analysen, Umstrukturierungen bei Ressourcenausfall usw. Das Vermessen von Ressourcen kann hier also bereits während der Planungsphase geschehen (z.B. wieviele Mitarbeiter werden an einem Teilprojekt arbeiten, zu wieviel Prozent ist eine Ressource ausgelastet, usw.). Für die Einbettung des Werkzeugs in ein Softwareprojekt liefert Micro Planning ein Vorgehensmodell mit, das wie folgt strukturiert ist:

1. *Projektdateneingabe*, d.h. sowohl die zur Verfügung stehenden Ressourcen als auch die möglichst feingranulare funktionale Zerlegung des Projekts (dort: *Work Breakdown Structure*) werden hier eingegeben. Jede Teilaufgabe wird grob bzgl. ihres Aufwands und ihres Ressourcenverbrauchs geschätzt.
2. *Projektplanung*, d.h. unter Berücksichtigung von Abhängigkeiten zwischen den (Teil-) Aufgaben wird ein grober Projektplan erstellt. Notwendige Ressourcen werden hier noch als beliebig verfügbar angesehen. Diese in Form von Netzwerkdiagrammen (*PERT-charts*) dargestellten Informationen beinhalten darüber hinaus noch Ressourcenbedarf und mögliche Zeitfenster, in denen die Aktivitäten verschoben werden können, ohne daß sich die Gesamtzeit des Projekts verändert.
3. *Ressourcenoptimierung*, d.h. das möglichst optimale Abbilden der zur Verfügung stehenden Ressourcen auf den Projektplan. Mittels verschiedenartiger Ressourcen-Histogramme können verschiedene Aspekte der Ressourcenmessung anschaulich visualisiert werden. Somit sind bereits während der Planung Ressourcenarmut und Ressourcenüberfluß zu erkennen und können entsprechend korrigiert werden.
4. *Projektkonsens*, d.h. die Diskussion des Projektplans und der Ressourcenaufteilung innerhalb der Projektgruppe. Jeder Mitarbeiter überprüft die für ihn geplanten Teilaufgaben und die dafür zur Verfügung gestellten Ressourcen. Eventuelle Anpassungen können diskutiert und ggf. modifiziert werden.
5. *Projektdurchführung*, d.h. die Durchführung des im Projektkonsens abgestimmten Projektplans entsprechend der Ressourcenaufteilung. Abweichungen von der Planung werden sofort hervorgehoben und mögliche Lösungen für den jeweiligen Kontext vorgeschlagen.

Das eigentliche Vermessen findet hauptsächlich in Punkt 3) aufbauend auf den in Punkt 1) und 2) eingegebenen Daten statt. Verschiedene Formen der Visualisierung der Ressourcenvermessung erlauben einen Vergleich mit der Planung und ermöglichen ein zügiges Erkennen von Effektivitätsdefiziten beim Ressourceneinsatz.

Da Ressourcen häufig nicht losgelöst vom Prozeß, in dem sie verbraucht werden, und dem erstellten Produkt, für den sie eingesetzt werden, betrachtet werden können, existieren auch integrierte Werkzeuge, die die Meßwerte aller drei Entitätstypen verarbeiten. So ist z.B. die Dateneingabe des Werkzeugs *Slim-Metrics* [Heir98] in die drei Bereiche Ressourcendaten, Prozeßdaten und Produktdaten aufgeteilt. Nach der (teilweise manuellen) Dateneingabe und einer darauf aufbauenden intensiven Vermessung in der Datenanalyse ist es möglich, die eigenen Datensätze mit beliebigen anderen Daten (eigene oder aus einer mitgelieferten, mit anonymisierten Daten absolvierter Projekte anderer Firmen gefüllten Industriedatenbank) in verschiedenster Weise zu kontrastieren. Für diesen Zweck sind in das Analysetool eine Vielzahl von statistischen Techniken und Visualisierungsarten eingebaut, um die eigenen Projektdaten möglichst leicht mit anderen Projekten (und deren Daten) vergleichen zu können. Damit

ermöglicht Slim-Metrics die relative Einordnung des eigenen verwendeten Prozesses, der Effektivität der verwendeten Ressourcen und der Güte der erzeugten Produkte im Vergleich zu einer Vielzahl anderer Projekte und gibt aufgrund der feingranularen Datenhaltung zugleich konstruktive Hinweise, welche Parameter optimiert werden können.
Darüber hinaus bietet es – nach entsprechendem Aufbau einer eigenen Datenbank über verschiedene Versionen eines Produkts, Prozesses oder einer Ressource – die Möglichkeit der Trendanalyse (vgl. Abschnitt 3.3.5).

3.4.3 Beispiele von Softwareprodukt-Meßwerkzeugen

Die Grundstruktur von Softwareprodukt-Meßwerkzeugen läßt sich in Form eines Datenfluß-diagramms wie in Abbildung 9 darstellen:

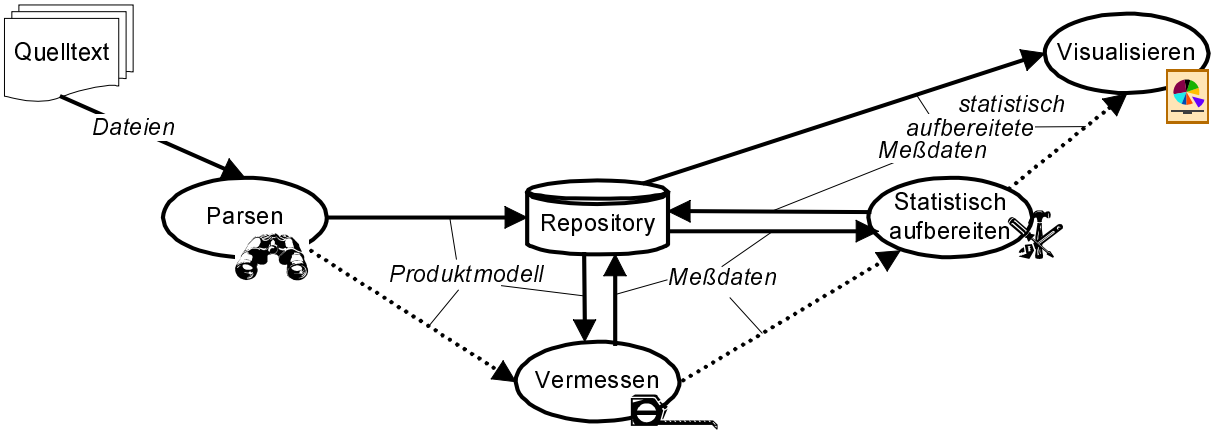


Abbildung 9: Aufbau eines Softwareprodukt-Meßwerkzeugs

Die einzelnen Prozesse haben dabei folgende Aufgaben

- Das *Parsen* dient der Erstellung des Produktmodells (vgl. Abschnitt 3.2). Dessen Umfang bestimmt die Art und Mächtigkeit der weiteren Analyse, da die damit vorgenommene Abstraktion (vgl. Abschnitt 3.2) bereits zielorientiert vorzunehmen ist. Das Produktmodell selbst wird häufig in ein *Repository* abgelegt, d.h. die technische Realisierung der Verwaltung der relevanten Entitäten und Relationen in Form einer Datenbank oder unter direkter Verwendung des Dateisystems. Ist das Produktmodell sehr einfach (z.B. für eine ausschließliche Betrachtung des Maßes LOC), so muß das Repository nicht explizit angelegt werden, sondern fließt dann unmittelbar als Strom in die Folgeprozesse (vgl. gepunktete Datenflüsse in Abbildung 9).
- Das *Vermessen* bestimmt die verschiedenen Größen innerhalb des Produktmodells. Welche Größen bestimmt werden hängt vom jeweilig verwendeten, angepaßten Qualitätsmodell (vgl. Kapitel 2) oder vom Funktionsumfang des jeweiligen Vermessungswerkzeugs ab. Das Ergebnis der Vermessung sind Meßdaten, bestehend aus einer Menge von Meßobjekten und dazugehörigen Meßwerten. Das Vermessen selbst kann nicht losgelöst vom jeweilig verwendeten Produktmodell betrachtet werden, da Maße i.d.R. ein bestimmtes Produktmodell implizit voraussetzen und Inkompatibilitäten zwischen beiden zu unsinnigen Meßwerten führen können. Die Meßdaten werden i.d.R. wieder im Repository abgelegt, können aber bei entsprechend geringem Umfang auch direkt als Strom an die Folgeprozesse geschickt werden.
- Die *statistische Aufbereitung* der Meßdaten versucht, die große Menge von Meßdaten zielorientiert zu konzentrieren. Für diesen Zweck werden beliebige Kombinationen der oben genannten Techniken Statistik, Wertefilterung und Komponentenfilterung verwendet (vgl. Abschnitt 3.4). Das Ergebnis ist ein Konzentrat der ursprünglichen Meßdaten, das die

entsprechend der Zielvorgaben durchzuführende Interpretation der Meßwerte unterstützt. Auch diese Ergebnisse können für die weiteren Schritte in das Repository abgelegt werden.

- Das *Visualisieren* versucht, die konzentrierten Meßdaten derart darzustellen, daß eine möglichst effiziente und effektive Interpretation der Meßdaten und notwendigerweise auch eine einfache Identifizierung der den Meßwerten zugrundeliegenden Quelltextteile möglich ist.

Entsprechend dieser Grobarchitektur lassen sich Softwareprodukt-Meßwerkzeuge durch folgende acht Aspekte bestimmen:

1. *Art und Entwicklungsstand des Quelltextes*, auf dem die Quelltextparser arbeiten können (mögliche Arten sind z.B. C++ und JAVA-Quelltexte; mögliche Entwicklungsstände sind z.B. nicht compilierbar, compilierbar oder compilierbar und linkbar),
2. *Art des Produktmodells*, d.h. Umfang der geparsten Daten,
3. Menge der innerhalb des Vermessens *berechenbaren Maße* und deren Anpaß- und Erweiterbarkeit,
4. *Möglichkeiten der statistischen Aufbereitung* und deren Anpaßbarkeit und
5. *Art der Visualisierungen* und deren Anpaßbarkeit.

Die folgenden drei Aspekte beziehen sich auf die technische Realisierung der Softwareprodukt-Meßwerkzeuge:

6. *Produktmodellursprung*: Erstellt das Werkzeug selbst das Repository, d.h. liefert es seinen eigenen Parser mit oder baut es auf dem Produktmodell auf, das durch verschiedene Fremdparser erstellt werden kann? Der Vorteil der ersten Variante liegt in der dadurch gegebenen Möglichkeit, das Produktmodell und die verwendeten Maße kompatibel zueinander zu halten. Der Vorteil der zweiten Variante liegt in der Möglichkeit, auch spezialisierte Quelltexte in verschiedenen Sprachdialekten vermessen zu können, so lange das produzierte Produktmodell kompatibel zu den verwendeten Maßen ist. Darüber hinaus ist diese Variante häufig schneller, da ein separates Parsen vom Softwareprodukt-Meßwerkzeug durch die Wiederverwendung des innerhalb von Entwicklungswerkzeugen häufig implizit gehaltenen Repositories entfällt.
7. *Entwicklungswerkzeug-Integration*: Der Erfolg von Softwarevermessung hängt stark vom notwendigen Aufwand ab, bestehende Daten zu vermessen und die die Meßwerte verursachenden Quelltextteile wieder in den ursprünglichen Daten zu identifizieren. Um beide Kriterien zu optimieren, ist es hilfreich, das Softwareprodukt-Meßwerkzeug als *Plug-In* zu einem Entwicklungswerkzeug zu konstruieren, da dadurch das Vermessen für den Entwickler aus der ihm bekannten Umgebung ohne viel zusätzlichen Aufwand gestartet werden kann. Auch die Identifikation von vermessenen Entitäten innerhalb der bekannten Umgebung ist effizienter als die separate Darstellung in Fremdwerkzeugen.
8. *Meßdatenhaltung*: Häufig ist es sinnvoll, sowohl die Meßdaten als auch die statistisch aufbereiteten Meßdaten in einem expliziten Repository zu halten. Wird diese Archivierung konsequent über mehrere Varianten eines Produkts durchgeführt, so wird dadurch die Möglichkeit einer Trendanalyse (vgl. Abschnitt 3.3.4) gegeben. Das Softwareprodukt-Meßwerkzeug kann diese Meßdatenhaltung in verschiedenen Ausprägungen unterstützen, die von einfachen Listenausdrucken für manuelle Trendanalysen bis hin zu eigenen Meßdatenrepositories für eine bei entsprechenden Daten für die Verfolgung einzelner Komponenten über verschiedene Versionen hinweg mögliche vollautomatische Trendanalyse reichen. Mit der Meßdatenhaltung eng verknüpft ist ebenfalls die für das Vermessen notwendige Zeit, da bei geeigneter Meßdatenhaltung minimale Änderungen am Quelltext durch inkrementelle Algorithmen nicht zu einer kompletten Neuvermessung führen müssen.

Ein solches inkrementelles Vorgehen muß allerdings durch entsprechende *Delta-Techniken* (vgl. z.B. [Alde93], S. 34/7ff) innerhalb des Produktmodells unterstützt werden.

Im folgenden soll entsprechend dieser acht Aspekte ein kommerzielles Softwareprodukt-Meßwerkzeug vorgestellt werden (Abschnitt 3.4.4), bevor in Abschnitt 3.4.5 ein eigenes, im Rahmen dieser Arbeit entwickeltes Werkzeug detailliert beschrieben wird. Die Vorstellung entlang dieser acht Aspekte hat folgende zwei Vorteile:

1. Durch die aufgeführten Aspekte kann die Produktfunktionalität vollständig beschrieben werden, d.h. es existiert eine Beschreibungsstruktur, mit der jedes Softwareprodukt-Meßwerkzeug bzgl. der angebotenen Funktionalität effizient dargestellt werden kann.
2. Beschreibungen verschiedener, mittlerweile zahlreich existierender Softwareprodukt-Meßwerkzeuge entlang dieser aufgeführten Aspekte erleichtern den Vergleich zwischen den einzelnen Werkzeugen und ermöglichen damit die Selektion desjenigen Meßwerkzeugs, das innerhalb eines konkreten Kontexts den größten Nutzen verspricht.

3.4.4 Datrix

Datrix ist eine firmeninterne, aber ebenfalls käuflich erwerbbare Sammlung von Analyse-Werkzeugen des kanadischen Telekommunikationsproviders *Bell Canada*. Dort wird Datrix primär zur qualitativen Bewertung von auswärtig erstellten Softwarepaketen entsprechend der ISO 9126 verwendet (vgl. [MaLa96], [Lagu97]), um die Qualität von, innerhalb Bell Canada häufig durchgeführten, Outsourcing-Projekten zu überprüfen und zu gewährleisten (vgl. Abschnitt 3.3.1). Im folgenden wird Datrix entsprechend der acht einzelnen Aspekte der in Abschnitt 3.4.3 aufgezeigten Architektur beschrieben:

1. *Art und Entwicklungsstand des Quelltextes*: Datrix ist in der Lage, Produktmodelle von Quelltexten zu erstellen, die in C++, C (Komponente *dxparscpp*) oder Java (Komponente *dxparsjava*) geschrieben sind. Die Quellen müssen dabei vollständig und kompilierbar sein.
2. *Art des Produktmodells*: Das Produktmodell selbst wird innerhalb von Datrix als *Intermediate Representation Language* bezeichnet und enthält den Datendeklarationsgraph, den Kontrollflußgraph und den Datenflußgraph. Die in der *Intermediate Representation Language* gehaltenen Informationen, die zusammen das Produktmodell ergeben, und die daraus ableitbaren Graphen variieren je nach Programmiersprache. Für die statische, objektorientierte Struktur von C++- oder JAVA-Quelltext wird z.B. ein Produktteilmodell verwendet, wie es in Abbildung 10 in der Klassendiagrammnotation der *Unified Modelling Language* (*UML*) dargestellt ist (vgl. [MaLa96]).

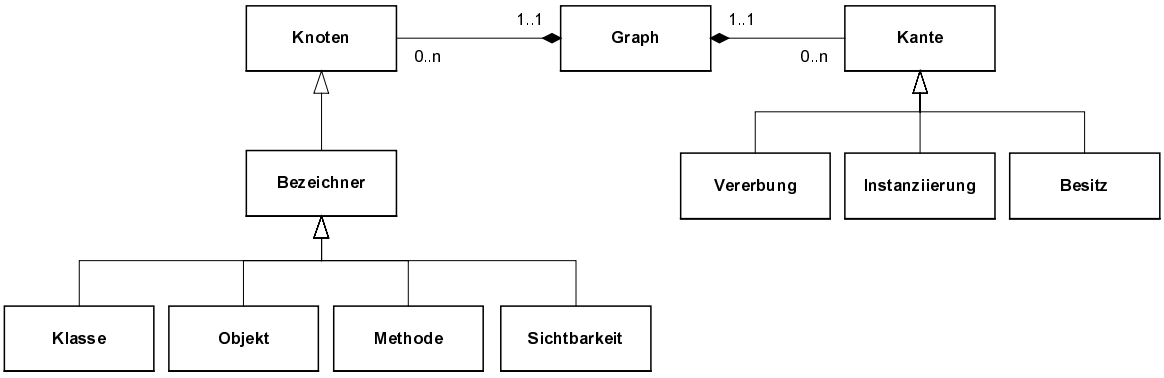


Abbildung 10: Datrix-Produktmodell für C++/JAVA (vgl. [MaLa96])

Das als Graph konzipierte Produktteilmodell beschreibt Klassen, Objekte, Methoden und Sichtbarkeiten als Knoten, während als Kanten die Beziehungen Vererbung, Instanziierung und Besitz betrachtet werden.
Das Repository, das das Produktmodell aufnimmt, ist innerhalb von Datrix als Textfile implementiert.

3. *Berechenbare Maße*: Je nach verwendetem Produktmodell finden entsprechende Meßprogramme Anwendung: Für Produktmodelle, die ihren Ursprung in C++ und C-Programmen haben, wird das Meßwerkzeug *dxmetcpp* verwendet; für Produktmodelle von JAVA-Quelltexten existiert das Werkzeug *dxmetjava*. Die Ausgabe der Meßwerkzeuge sind textuelle Meßwertaufzählungen, die in entsprechende Dateien umgeleitet werden können, um von entsprechenden Aufbereitern und Visualisierern gelesen werden zu können.
   Die Menge der innerhalb von Datrix berechenbaren Maße ist fix und läßt weder eine Maßmodifikation noch eine Maßneudefinition (z.B. aufgrund spezieller Qualitätsmodelle) zu. Der mitgelieferte Maßkatalog umfaßt über 60 Maße und ist innerhalb von Datrix wie folgt klassifiziert:
   - *Klassenmaße*, die Klassen in objektorientierten Produktmodellen vermessen (z.B. Anzahl öffentlicher Attribute, *ClaAttPubNbr*, oder Länge des Klassennamens, *ClaNamLen*),
   - *Routinenmaße*, die Funktionen bzw. Methoden von C-, C++- und Java-Programmen vermessen (z.B. Anzahl von Kommentaren innerhalb eines Funktionsrumpfes, *RtnComNbr*, oder Anzahl von Befehlen innerhalb eines Funktionsrumpfes, *RtnStmNbr*), und
   - *Dateimaße*, die Dateien als ganzes vermessen (z.B. alle Halstead-Maße [Hals77] wie Halstead-Länge, *FilHalLen*, oder Halstead-Umfang, *FilHalVol*).
4. Die *Möglichkeiten der statistischen Aufbereitung* der Meßwerte werden nicht durch Datrix-eigene Werkzeuge abgedeckt, sondern werden Standardanwendungen überlassen. Dokumentierte statistische Techniken im Rahmen von Datrix-Experimenten umfassen Funktionen wie Mittelwert, Median, Standardabweichung, Wertebereich, Minimum, Maximum, Filterwerte, Verteilungen und Klassifizierungen ([MaLa96], [Lagu97]).
5. Die *Visualisierung* wird ebenfalls nicht durch Datrix-eigene Werkzeuge durchgeführt. Dokumentierte Visualisierungen im Rahmen von Datrix-Experimenten umfassen Balkendiagramme, Verteilungsdiagramme, Kurvendiagramme und Scatterplots.

Die Aspekte der technischen Realisierung von Datrix lassen sich wie folgt beschreiben:

6. *Produktmodellursprung*: Datrix besitzt seine eigenen Parser für die Erstellung des Produktmodells. Es ist nicht vorgesehen, Fremdparser einbinden zu können.
7. *Entwicklungswerkzeug-Integration*: Datrix ist als Stand-Alone-Applikation konzipiert, d.h. es existiert keine Datrix enthaltende Entwicklungsumgebung, von der aus der Meßvorgang initiiert werden kann und mittels der die Ergebnisse auf Quelltextteile zurück projiziert werden können.
8. *Meßdatenhaltung*: Einzige Möglichkeit der Archivierung der Meßwerte ist das Umleiten von textuellen Ausgaben in eine Datei.

Innerhalb des Internets ist Datrix durch die Vermessung des *Netscape*® Browsers, der größtenteils in C geschrieben ist, bekannt geworden. Datrix stellt eher eine für den konkreten Kontext noch zusammenzustellende, Shell-orientierte Werkzeugsammlung als ein kompaktes, einfach zu installierendes und anzuwendendes Meßwerkzeug dar. Besonders das Fehlen von Möglichkeiten zur Anpassung der Menge verwendeter Maße, die fehlende Unterstützung, diese in Abhängigkeit eines spezifischen Qualitätsmodells zu interpretieren und die ungenügende Unterstützung einer entsprechenden Meßwertaufbereitung stellen zur Zeit noch ein wesentliches Problem von Datrix für dessen Anwendung zur Unterstützung von Qualitätssicherungsmaßnahmen dar. Da es aber

nach wie vor gepflegt wird und mittlerweile für eine Vielzahl von Betriebssystemen existiert, ist langfristig sicherlich der Ausbau der Kommerzialisierung von Datrix angestrebt.

3.5 Crocodile

Crocodile ist ein im Rahmen dieser Arbeit am Lehrstuhl *Software-Systemtechnik* der *Technischen Universität Cottbus* entwickeltes Softwareprodukt-Meßwerkzeug, das zudem als Public-Domain-Software via Internet verfügbar ist [SiLe98]. Crocodile selbst ist dabei nicht nur als Produkt konzipiert, das möglichst einfach zu installieren ist, möglichst wenig äußere Softwareanforderungen stellt und auf möglichst vielen Betriebssystemen lauffähig ist, sondern es ist ebenfalls als diejenige Umgebung konzipiert, in der die in dieser Arbeit vorgestellten Konzepte praktisch umgesetzt werden, um dann – nach angemessenem Testen – sukzessive ebenfalls in das ausgelieferte Produkt übernommen werden zu können. Crocodile stellt somit eine zuverlässige Implementierung der Werkzeugunterstützung der in dieser Arbeit vorgestellten Konzepte dar und wurde daher auch für die im Rahmen dieser Arbeit durchgeführten Projekte erfolgreich verwendet.
In diesem Abschnitt soll derjenige Teil von Crocodile vorgestellt werden, der im Rahmen der exemplarischen Vorstellung klassischer Softwareprodukt-Meßwerkzeuge relevant ist. Darüber hinausgehende Funktionalität, die neuere Konzepte implementiert, wird in späteren Kapiteln ergänzt.
Crocodile wird im folgenden – wie bereits bei Datrix – entlang der oben vorgestellten acht Aspekte charakterisiert. Im Anschluß werden darüber hinaus die praktischen Erfahrungen aufgelistet, die im Rahmen dieser Arbeit bzgl. der Anwendung der klassischen Softwareproduktvermessung mit Crocodile gemacht wurden.

1. *Art und Entwicklungsstand des Quelltextextes*: Crocodile ist in der Lage, C++- und Java-Quelltexte in ein Produktmodell zu überführen. Aufgrund dessen hoher Abstraktion und der Einbindung von Fuzzy-Parsern ist es darüber hinaus möglich, nicht compilierbaren Quelltext, sowie Programmfragmente (sogenannte *Programmskelette*), die lediglich aus den Deklarationen bestehen, zu vermessen. Damit wird das Vermessen von bereits in der Analyse- und Entwurfsphase erstellten Klassendiagrammen unterstützt, da viele Entwicklungswerkzeuge in der Lage sind, aufbauend auf der grafischen Klassendiagrammnotation programmiersprachenabhängige Programmskelette zu generieren. Das Vermessen von reinen Entwurfsdokumenten ermöglicht daher ein sehr frühes Erkennen von qualitätsverletzenden Eigenschaften, die zu dem Zeitpunkt noch relativ kostengünstig behoben werden können.
2. *Art des Produktmodells*: Das Produktmodell innerhalb von Crocodile ist speziell für objektorientierte Systeme entwickelt worden und läßt sich als Klassendiagramm wie in Abbildung 11 darstellen. Ein zu vermessendes System zerfällt demnach in mehrere Subsysteme, in denen sich mehrere Dateien befinden können. Dateien sind jeweils eindeutiger Deklarationsort der Klassen, der Attribute und der Methoden. Klassen selbst können voneinander erben und können mehrere Methoden und Attribute besitzen. Darüber hinaus werden im Crocodile-Datenmodell Benutztbeziehungen zwischen Methoden und zwischen Methoden und Attributen gehalten.

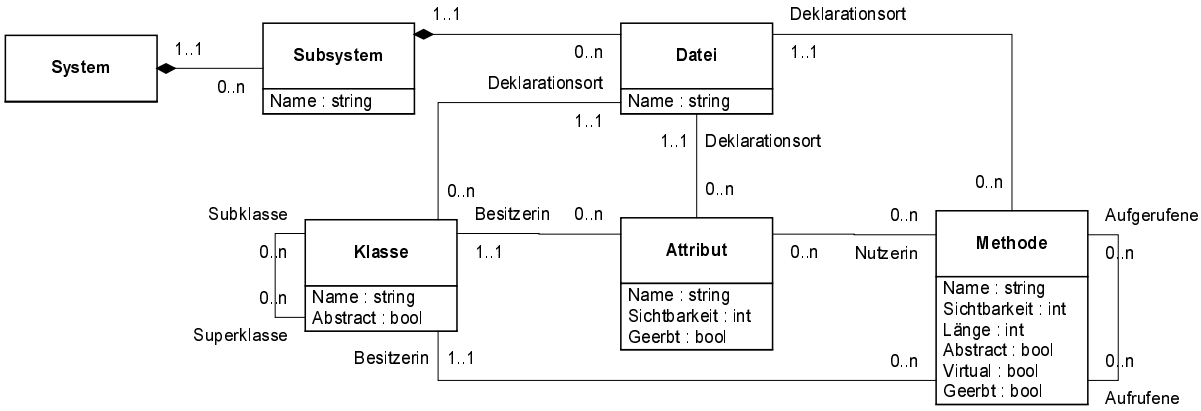


Abbildung 11: Crocodile-Produktmodell

Eine Besonderheit stellen die Attribute „Geerbt“ bei den Attributen und Methoden dar: Mit ihnen ist es möglich, Sichten auf das System zu erzeugen, innerhalb derer jede Klasse vollständig, d.h. inkl. aller ihrer geerbten Funktionalität dargestellt werden kann. Dieses Konzept ist detailliert in Kapitel 8.2. erläutert.

3. *Berechenbare Maße:* Ein wesentlicher Vorteil von Crocodile ist die vollständige Konfigurierbarkeit der zu berechnenden Maße und deren Integration in ein ebenfalls anpaßbares Qualitätsmodell. Für diesen Zweck bietet Crocodile eine SQL-ähnliche *Metrics-Definition-Language (MDL)*, mit deren Hilfe es möglich ist, Maße aufbauend auf dem Produktmodell zu definieren. Die MDL klassifiziert die definierbaren Maße entsprechend der vermessenen Entitäten in
   - *Subsystemmaße*, d.h. Maße, die Charakteristika von Subsystemen vermessen (z.B. Kopplung zu anderen Subsystemen),
   - *Dateimaße*, d.h. Maße, die Charakteristika von Dateien vermessen (z.B. wieviele Klassen in einer Datei deklariert sind); darüber hinaus erlauben Dateimaße den direkten Zugriff auf die Datei selbst, um über das Produktmodell hinausgehende Informationen zu erhalten, die i.d.R. nicht von Standardparsern ermittelt werden (z.B. Anzahl von Brutto-Lines of Code, Anzahl von Kommentarzeilen o.ä.). Diese Produktmodellumgehung wurde aufgrund häufiger Anfragen nach klassischen, nicht typisch objektorientierten Maßen eingeführt.
   - *Klassenmaße*, d.h. Maße, die Charakteristika von Klassen vermessen (z.B. Kopplung zwischen Klassen, Anzahl der Methoden/Attribute usw.).
   - *Methodenmaße*, d.h. Maße, die Charakteristika von Methoden vermessen (z.B. Anzahl der Benutztbeziehungen zu anderen Klassen oder Attributen) und
   - *Attributmaße*, d.h. Maße, die Charakteristika von Attributen vermessen (z.B. Anzahl der Benutztbeziehungen von Methoden).

   Darüber hinaus erlaubt die MDL die Erstellung von anpaßbaren Qualitätsmodellen, an deren Knoten mit der höchsten Verfeinerungsstufe die definierten Maße integriert werden können, was eine spätere Interpretation der vermessenen Werte erleichtert.
4. *Möglichkeiten der statistischen Aufbereitung*: Crocodile selbst stellt folgende Techniken zur Verfügung, um die erzeugte Datenmenge zu konzentrieren:
   - *Statistische Funktionen*: Crocodile bietet innerhalb der MDL ein Grundrepertoire verschiedener Operatoren auf die Meßwerte an, die es erlauben, einfache statistische Berechnungen wie arithmetisches Mittel durchzuführen. Eine häufig verwendete statistische Funktion ist die Kombination mehrere Maße zu einem neuen Maß (vgl. [SiRuLe00]).

- *Komponentenfilterung*: Innerhalb Crocodile's existieren zwei Möglichkeiten, die Komponenten, für die Meßwerte bestimmt werden sollen, zu selektieren: Bereits zu Beginn der Produktmodellerstellung kann auf Subsystemebene entschieden werden, welche Subsysteme in das Produktmodell überführt werden sollen. In einem zweiten Schritt ist es darüber hinaus ebenfalls möglich, innerhalb eines Produktmodells diejenigen Komponenten zu selektieren, deren Meßwerte von Interesse sind. Diese zweifache Komponentenfilterung ermöglicht das explorative Vermessen unterschiedlicher Systemteile eines (größeren) Produktmodells und erlaubt spezielle Veränderungen des Produktmodells, die das Konzept der Vererbung innerhalb objektorientierter Systeme berücksichtigen (vgl. Kapitel 8.2 und [SiBe00]).
- *Wertefilter*: Crocodile bietet die Möglichkeit, bereits in der MDL Wertefilter anzugeben, deren Verletzung Auswirkungen innerhalb des Qualitätsmodells hat. Mögliche Filter sind absolute Werte (z.B. alle Werte>25), relative Werte (z.B. die größten 25 Werte), absolute offene Intervalle (z.B. alle Werte <5 oder >10) und absolute geschlossene Intervalle (z.B. alle Werte >5 und <10).

Durch einen speziellen *Microsoft-Excel*®-Konverter für die Meßwerte ist es ebenfalls möglich, dessen umfangreiche statistische Fähigkeiten zu verwenden (z.B. für die Durchführung von Häufigkeitsanalysen).

5. *Art der Visualisierungen*: Die visuelle Aufbereitung der Meßdaten innerhalb von Crocodile beschränkt sich auf ein Minimum (scrollbare Listen, Extremwertdarstellung mittels Balkendiagrammen und Qualitätsmodellvisualisierung inkl. Pfadhervorhebung, Wertefilterverletzungen), da durch den Excel®-Konverter dessen umfangreiche Visualisierungsmöglichkeiten zur Verfügung stehen, die durch eigens dafür entwickelte Visualisierungsvorlagen ebenfalls semi-automatisch angewendet werden können. Dokumentierte Visualisierungsarten sind parametrisierbare zwei- und dreidimensionale Balkendiagramme, Verteilungsdiagramme und Kurvendiagramme [SiRuLe00].
   Eine wichtige Forderung der Visualisierung – die einfache Identifikation der einem Meßwert zugrundeliegenden Entität innerhalb des Projekts – wird durch die Einbettung von Crocodile in bestehende Softwareentwicklungsumgebungen erreicht (vgl. Punkt 6 - 8). So wird bei Betrachtung eines konkreten Meßwerts auf Wunsch die entsprechende Entität innerhalb der Softwareentwicklungsumgebung angezeigt.

Bzgl. der technischen Realisierung ist ein wesentlicher Unterschied von Crocodile zu anderen existierenden Softwareprodukt-Meßwerkzeugen dessen Softwareumgebung: Es ist als Plug-In zu verschiedenen Entwicklungsumgebungen konzipiert (z.B. *SNiFF+*, vgl. [Pfei97], oder *Source Explorer* , vgl. [Died99]) und hält sein eigenes Meßdatenrepository in Form einer relationalen Datenbank. Sowohl zur Entwicklungsumgebung als auch zur Datenbank existieren Schnittstellen, die von verschiedenen konkreten Produkten implementiert werden können. Für die Aspekte der technischen Realisierung folgt daraus:

6. *Produktmodellursprung*: Crocodile besitzt keine eigenen Parser, sondern liest das von Entwicklungsumgebungen selbst gehaltene Repository aus und konvertiert die Daten in das eigene Produktmodell. Crocodile kann daher ohne separate Entwicklungsumgebung kein Produktmodell erstellen. Die Vermessung selbst ist allerdings mit einer Entwicklungsumgebung sehr einfach, da kein separates Parsen notwendig ist und auch keine neuen Probleme mit unterschiedlichen Sprachdialekten auftreten, da diese – wenn vorhanden – i.d.R. bereits während der Einführung des Entwicklungswerkzeugs behoben wurden.

7. *Entwicklungswerkzeug-Integration*: Aufgrund des Charakters als Plug-In ist die gesamte Funktionalität von *Crocodile* in die Entwicklungsumgebung integriert, d.h. dort werden entsprechende Menüpunkte hinzugefügt. Dadurch wird ein sehr einfacher Umgang mit dem Meßwerkzeug erreicht, da die für den Entwickler gewohnte Umgebung für ein Vermessen nicht verlassen werden muß. Ebenfalls die Identifikation von vermessenen Entitäten (vgl. Punkt 5) geschieht innerhalb der bekannten Umgebung.
8. *Meßdatenhaltung*: Nicht nur das Produktmodell, sondern ebenfalls die errechneten Meßwerte werden in einer Datenbank abgelegt. Das Vermessen selbst muß daher nur einmal ausgeführt werden und kann anschließend beliebig exploriert werden. Ein wesentlicher Vorteil des Meßdatenrepositories ist allerdings die Möglichkeit der Trendanalyse (vgl. Abschnitt 3.3.4): Durch das Anlegen mehrerer Datensätze für verschiedene Versionen eines Produkts und die automatische Identifikation von gleichen Entitäten über die verschiedenen Versionen hinweg sind meßwertbasierte Trends errechenbar (vgl. [SiRuLe00]). Diese können für alle in allen Versionen errechneten Maße automatisiert und entsprechend visualisiert werden.
   Ein Vorteil der expliziten Produktmodellhaltung ist die Möglichkeit der externen Qualitätsbeurteilung ohne Notwendigkeit der Quelltextfreigabe: Wird das Produktmodell innerhalb des das Produkt erstellenden Unternehmens erzeugt, so kann diese Abstraktion ohne dem darunter liegenden Quelltext an externe Beurteilungsgruppen geschickt werden. Diese Möglichkeit ist speziell für Universitäten interessant, die empirische Untersuchungen mit Softwaremaßen für große Systeme durchführen wollen, da kommerzielle Unternehmen ihnen aus Sicherheitsgründen nur selten den Quelltext zukommen lassen (vgl. Kapitel 9).

Crocodile selbst wurde bzgl. dieser Kernfunktionalität erfolgreich für folgende Meßprozesse eingesetzt (vgl. Abschnitt 3.3):
- Outsourcing-Beurteilung im Rahmen einer Zusammenarbeit mit *ABB* (vgl.[SiRuKö98]),
- Vorbereitung eines Reviews im Rahmen einer Zusammenarbeit mit dem *IBL Ingenieurbüro* und im Rahmen eines Studentenexperiments (vgl. [SiRuKö98]),
- Überarbeitung eines Programms im Rahmen eines Studentenexperiments (vgl. [SiRuKö99a], [SiLö00]) und
- Trendanalyse im Rahmen einer Zusammenarbeit mit *Siemens* (vgl. [SiRuLe00]).

3.6 Probleme beim Produktmaßeinsatz

Trotz umfangreicher Arbeiten auf dem Gebiet der Softwareproduktvermessung, trotz existierender Werkzeuge und trotz der allgemein anerkannten Notwendigkeit, möglichst frühzeitig Qualitätsdefizite mit möglichst minimalem Aufwand aufzudecken, hat sich der Einsatz von Produktmaßen noch lange nicht innerhalb der Softwarebearbeitung etabliert. Die Probleme, die diese Diskrepanz zwischen Einsicht der Notwendigkeit und tatsächlicher Anwendung möglicher Lösungen begründen, können, entsprechend eigener im Rahmen dieser Arbeit gemachter Erfahrungen klassifiziert werden in
- *allgemeine Probleme* der meßbasierten Qualitätsanalyse, die prinzipiell bei jedem Einführen neuer Methoden und Werkzeuge innerhalb des Software-Engineering auftreten können und
- *spezifische Probleme* der meßbasierten Qualitätsanalyse, die durch spezielle Eigenschaften von Softwaremaßen, deren Umsetzung und der anvisierten Ziele gegeben sind.

Die Identifikation der ersten Problemklasse birgt den Vorteil, bereits bekannte Problembereiche (und eventuell dafür bereits vorgeschlagene Lösungen) für die Analyse der Probleme (und deren eventuelle Lösungen) beim Einsatz von Produktvermessung heranziehen zu können. Dieser Transfer wird detailliert in Abschnitt 3.6.1 durchgeführt, wobei sowohl eigene als auch in der Literatur berichtete Erfahrungen beim Einsatz von Produktvermessung berücksichtigt werden.

Der durch diese Problembeschreibungen aufgespannte Problembereich ist sehr weit und umfaßt sowohl technische, organisatorische als auch psychologische Bereiche.
Die Beschreibung der zweiten Problemklasse in Abschnitt 3.6.2 zeigt für die meßbasierte Qualitätsanalyse spezifische Probleme. Diese haben i.d.R. direkte Auswirkungen auf die allgemeinen Probleme, bieten allerdings aufgrund ihres spezifischeren Charakters konkretere Ansatzpunkte für die Lösung der Probleme. In Abschnitt 3.6.3 wird darauf aufbauend die weitere Zielsetzung dieser Arbeit beschrieben, in der es um die Behebung wesentlicher spezifischer Probleme beim Produktmaßeinsatz geht und damit auch um die Behebung einiger allgemeiner Probleme bei der Einführung von Softwaremaßen.

3.6.1 Allgemeine Probleme beim Produktmaßeinsatz

Die allgemeinen Probleme beim Einführen neuer Methoden und Werkzeuge innerhalb des Software-Engineering können in die kritischen Faktoren *Vorgehen, Management, Promotoren, Betroffene, Ausbildung und Unterstützung, Pilotprojekte* und *Werkzeuge* unterteilt werden ([Sieg84], [Wall90]). Jeder Faktor wird im folgenden jeweils kurz vorgestellt (linke Spalte) und bzgl. des Problemfelds des Produktmaßeinsatzes anhand vieler, jeweils im Rahmen dieser Arbeit durchgeführten Projekte konkretisiert (rechte Spalte). Viele bereits existierenden Problemberichte aus der Literatur beim Einsatz von Produktmaßen sind diesen Faktoren ebenfalls gut zuzuordnen, so daß sie dort jeweils entsprechend eingeordnet und vorgestellt werden.

Vorgehen

Dieser Bereich umfaßt Probleme, die den Prozeß beschreiben, innerhalb dessen neue Methoden und Werkzeuge eingeführt werden. Typische Schwachstellen dieser Problemklasse sind (vgl. [Sieg84], [Wall90])

Risiken des Vorgehens zur Einführung neuer Techniken...	**... konkretisiert für die Technik Softwareproduktvermessung**
Eine fehlende Vorgabe realistischer Ziele und des erhofften Nutzens der Neueinführung, die zu einer überhöhten Erwartungshaltung beim Management und bei den Entwicklern führen kann.	Aus der Unterschiedlichkeit der Produktqualitätsvorstellungen der an der Einführung beteiligten Personengruppen (vgl. Kapitel 2.2) ergeben sich zwangsläufig unterschiedliche Ziele der neuen Methoden und Werkzeuge. Werden diese Unterschiede nicht thematisiert, um einen groben Konsens zu erreichen, ist das Erreichen der unterschiedlichen Einzelziele und damit die gesamte Neueinführung gefährdet. Darüber hinaus besteht gegenüber den Produktmaßen häufig in allen Personengruppen eine überhöhte Erwartungshaltung, da Messen als etwas tagtägliches bewährt zu sein scheint. Dies gilt speziell für die Ebene des Managements, das den Umgang mit Zahlen gewohnt ist und daher häufig jede für sie interessante Eigenschaft direkt vermessen möchte (z.B. Produktivität der Programmierer, Qualität des Produkts etc.): „*Häufige Ursachen des Scheiterns von Metrikprogrammen sind übertriebene technische Erwartungen, vor allem bezüglich Genauigkeit und Automatisierbarkeit* “ ([Rudo94], S. 35).

Eine ungenaue Festlegung der Einsatzumgebung der neuen Methoden und Techniken.	Aufgrund des tagtäglichen Umgangs mit Maßen wird die Notwendigkeit präziser Vorgaben für einen Transfer dieses Wissens auf den Bereich der Softwarevermessung häufig übersehen. Absolut notwendige Informationen wie z.B. die Art des Einsatzes (vgl. Abschnitt 3.3), präzise Definition selektierter Maße (vgl. Kapitel 3.2. und 4) oder Vorgaben zur Interpretation der Meßergebnisse fehlen häufig völlig.
Eine fehlende Rückkopplung der Erfahrungen mit der neuen Technik durch eine möglichst objektive Demonstration des Erfolgs oder Mißerfolgs an alle an den Neuerungen beteiligten Personen.	Maße werden häufig als reine Pflicht zur Meßdatenerzeugung aufgefaßt, deren Sinn dem jeweiligen Entwickler oder dem mittleren Management nicht klar ist. Dies gilt speziell für unternehmensweit angeordnete Vermessungen, deren Motivation häufig lediglich der Versuch ist, die in anderen Bereichen häufig gegebene Möglichkeit der einfachen Ermittlung von Daten wie Produktivität, Robustheit oder Verschleiß auf den Bereich der Softwarebearbeitung zu übertragen. Die daher fehlende Motivation innerhalb der „Meßexekutive“ (d.h. die Personen, die die Softwareproduktvermessung durchführen) für eine Untersuchung des Erfolgs der neuen Techniken begründet mangelnde Feedback-Loops, die projektspezifische Anpassungen ermöglichen würden, und damit den Erfolg der Neuerungen verbessern könnten.

Management
Dieser Bereich umfaßt Probleme, die die organisatorische Durchsetzung des Prozesses beschreiben, innerhalb dessen neue Methoden und Werkzeuge eingeführt werden sollen. Typische Schwachstellen dieser Problemklasse sind (vgl. [Sieg84], [Wall90]):

Risiken des Managements beim Einführen neuer Techniken...	**... konkretisiert für die Technik Softwareproduktvermessung**
Mangelnde finanzielle, organisatorische, technische und ideelle Voraussetzungen für die Neuerungen.	Die Neueinführung von Softwarevermessung und die damit notwendige Entwicklung eines angepaßten Qualitätsmodells geschieht iterativ, d.h. speziell die ersten Durchläufe werden i.d.R. nur suboptimale Ergebnisse liefern. Die Mächtigkeit von Maßen kann nur erschlossen werden, wenn im Vorfeld entsprechende Voraussetzungen geschaffen werden, die Freiraum für die Planung, Durchführung und anschließende Überarbeitung lassen: „*First, you need management support* “ ([Pigo96], S. 217). Solche Voraussetzungen sind z.B. • genügend Zeit (und z.B. nicht die Einführung von Softwarevermessung bei Termindruck für eine Produktauslieferung), • eine entsprechende Unternehmenskultur, die qualitativ hochwertige Software anstrebt und in der Kritik (z.B. an Teilen der Softwarevermessung) möglich und erwünscht ist, oder auch • die Integration der Softwarevermessung in die Softwareentwicklung anstelle einer bewußt extern gehaltenen Qualitätssicherungsgruppe.

Mangelnde Konsequenz bei der Überprüfung der Einhaltung einer erarbeiteten Einführungsstrategie	Der iterative Charakter der Meßprogrammeinführung verlangt ebenfalls eine konsequente Überprüfung der Einhaltung der erarbeiteten Verbesserungsvorschläge; übertragen auf die Feedback-Loops erfordert dies die Feststellung der Konsistenz zwischen der Plan- und der Do-Phase mittels der Check-Phase. Geschieht dies nicht in regelmäßigen Abständen, ist die Feedback-Loop gestört und die weitere Entwicklung der Meßeinführung nicht mehr zielgerichtet.

Promotoren
In diesem Bereich wird die Notwendigkeit der Motivation für eine neue Technik behandelt. Für diesen Zweck sind entsprechende Ressourcen (z.B. in Form von Promotoren) einzusetzen. Typische Schwachstellen sind (vgl. [Sieg84], [Wall90]):

Risiken der Motivation zur Einführung neuer Techniken...	**... konkretisiert für die Technik Softwareproduktvermessung**
Mangelnde Überzeugungskraft über die verschiedenen Unternehmensebenen hinweg	Um Softwarevermessung erfolgreich einsetzen zu können, müssen alle Unternehmensebenen involviert sein, da sowohl vom Management entsprechende Voraussetzungen geschaffen (s.o.), als auch die Meßexekutive motiviert sein müssen (s.o.). Eine ausschließlich von einer Ebene ausgehende Meßmotivation wird daher i.d.R. nicht zum gewünschten Erfolg führen: „*You must make sure that all of the people concerned, from upperlevel management to entry-level employees, understand the use of the measures to satisfy the goals of the company* [...] “ ([Pigo96], S. 217). Selbst die Einführung einer absolut minimalen Softwareproduktvermessung ist gefährdet, wenn nicht alle Beteiligten entsprechend überzeugt sind: „*Measurement cannot just be introduced. It must be ‚sold‘* “ ([Scha94], S. 54).
Mangelnde Betreuung über die Startphase hinweg.	Das Problem der Nachhaltigkeit der eingeführten Techniken ist speziell im Bereich der Softwarevermessung häufig gegeben, da z.B. eine Vermessung zur Standardeinhaltung (vgl. Abschnitt 3.3.1) i.d.R. nur zu fixen Zeitpunkten durchgeführt wird (z.B. in Form der jährlich stattfindenden Überwachungsaudits und der alle drei Jahre stattfindenden Wiederholungsaudits für eine Zertifizierung entsprechend der ISO 9000; vgl. z.B. in [Petr98] oder [Balz98]). Fehlt eine permanente Überzeugungskraft, so wird das hinter einer solchen Norm stehende Ziel der Erstellung qualitativ höherwertiger Software i.d.R. nicht erreicht.

Die Betroffenen
In diesem Bereich werden psychologische Probleme innerhalb der untersten operativen Ebene eines Unternehmens aufgrund der Neueinführung von Techniken und Werkzeugen behandelt. Typische Fehler sind (vgl. [Sieg84], [Wall90]):

Psychologische Risiken bei den Betroffenen durch Einführen neuer Techniken...	... konkretisiert für die Technik Softwareproduktvermessung
Die mangelnde Sensibilität bzgl. eines häufig anzutreffenden großen Mißtrauens gegenüber neuen Techniken speziell bei älteren Mitarbeitern, die über Jahre hinweg eigene Techniken entwickelt haben.	Da Softwarebearbeitung auch ohne Vermessung funktioniert (und teilweise auch derart betrieben wird), bedarf eine Neueinführung dieser Technik einer sensiblen Vorgehensweise. So muß speziell bei Softwareproduktmaßen verdeutlicht werden, daß nicht der Entwickler, sondern das Programm vermessen wird: „*Don't allow anyone in your organization to use metrics to measure individuals* " ([Scha94], S. 57). Darüber hinaus muß den Betroffenen der auch für sie mögliche Vorteil von Vermessung verdeutlicht werden (vgl. z.B. Abschnitt 3.3.3). „*Employees must understand that the way they do their job right now is fine. Collecting metrics will enable them to do their job better* " ([Pigo96], S. 217).
Die Nichtwürdigung erreichter Leistungen mit bisherigen Mitteln.	Ebenfalls kritisch ist die extern betreute ad-hoc Einführung der Softwarevermessung, ohne den aktuellen Stand innerhalb der Organisation zu berücksichtigen. So kann der grundsätzliche Vorteil einer Vermessung bereits in Ansätzen mit allgemeinüblichen Werkzeugen wie `grep` oder `wc` verdeutlicht werden (z.B. um das Maß LOC zu bestimmen), bzw. - bei entsprechender Unternehmenskultur - erweitert werden. Allgemeiner gilt: „*Metrikprogramme haben immer dann gute Erfolgsaussichten, wenn sie im Unternehmen wie andere, bereits bestehende, Betriebsdatenerfassungssysteme aufgebaut und betrieben werden* " ([Rudo94], S. 35).

Ausbildung und Unterstützung

Unter diesem Aspekt werden Risiken subsumiert, die den für die Einführung der Neuerung notwendigen Kenntnisstand und dessen Erweiterung betreffen. Typische Fehler sind (vgl. [Sieg84], [Wall90]):

Risiken der Ausbildung und Unterstützung beim Einführen neuer Techniken...	... konkretisiert für die Technik Softwareproduktvermessung
Die mangelhafte Ausbildung über alle Organisationsebenen hinweg. Dies meint das theoretische Lernen relevanter Informationen sowie die sofortige praktische Umsetzung innerhalb des Teams.	Messen als etwas tagtägliches wird häufig als bekannt vorausgesetzt und erscheint damit häufig als nicht notwendig zu lernen. Allerdings bedürfen die dort häufig implizit gemachten Voraussetzungen ebenfalls des Transfers auf den Bereich der Software, um aussagekräftige Ergebnisse zu erhalten. Das Lehren der Theorie (vgl. z.B. Kapitel 4) genügt allerdings nicht, sondern bedarf konkreter Einbindungen in die verschiedenen Prozesse. Speziell die Relativierung von meßwertbasierten Aussagen durch einen mündigen Umgang mit Meßwerten muß geschult werden.

Es muß die Möglichkeit gegeben werden (z.B. durch Musterprojekte), eigene Erfahrungen – positive und negative – zu machen, in denen das theoretische Wissen der aktuellen Projektsituation (wie z.B. Termindruck, veränderte Aufgabenstellung) angepaßt werden kann.	Viele Probleme des Messens treten erst in der Praxis auf. So scheitern beispielsweise viele Ansätze aufgrund der Größe praxisrelevanter Anwendungen oder der für das Vermessen notwendigen Ressourcen. Darüber hinaus ist das in Handbüchern vorgestellte Messen sehr viel einfacher als das Vermessen realer, zum Teil selbst geschriebener Systeme, da vom letzteren zusätzliche, semantische Informationen existieren, die die Meßwertinterpretation beeinflussen: z.B. Vorgaben von außen (u.a. die Festlegung bestimmter Schnittstellen), personelle Eigenheiten (u.a. individuelle Vorlieben bestimmter Konstrukte) oder relevante Ereignisse (u.a. Abwesenheit bestimmter Spezialisten).

Pilotprojekte

Die Einführung neuer Techniken und Methoden muß an Pilotprojekten mit nicht mit Risiko behafteten Applikationen durchgeführt werden. Trotz dieser Einsicht, die zwangsläufig aus den Bereichen Vorgehen, Management, die Betroffenen und Ausbildung und Unterstützung folgert, liegt in der Durchführung dieser Pilotprojekte eine weitere Problemquelle (vgl. [Sieg84], [Wall90]):

Risiken von Pilotprojekten zur Einführung neuer Techniken...	**... konkretisiert für die Technik Softwareproduktvermessung**
Die Ergebnisse des Pilotprojekts müssen dokumentiert und konsequent ausgewertet werden. Dies umfaßt Vor- und Nachteile der eingeführten Technik.	Die Durchführung von Pilotprojekten zur Einführung von Softwarevermessung beschränkt sich häufig auf die technische Machbarkeit der Vermessung. Eine Dokumentation und konsequente Auswertung findet selten statt, da beides für Pilotprojekte auch selten sinnvoll ist und sich innere Qualität von Software – in diesem Fall dann für das Pilotprojekt – nur über längere Zeiträume lohnt und kurzfristig jede Technik für deren Erhöhung als Arbeits-Mehraufwand betrachtet werden kann. Ein sinnvolles Pilotprojekt sollte also die reine Vermessung betrachten (vgl. Abschnitt 3.2), deren sinnvolle Anwendungen (vgl. Abschnitt 3.3) und letztendlich eine Untersuchung, ob die für das Unternehmen relevante Gesamtqualität (vgl. Kapitel 2) verbessert werden konnte.

Die Beteiligten des Pilotprojekts sollten anschließend als Know-How-Träger auf die relevanten Projekte verteilt werden. Dies fördert die Motivation (z.B. als Promotoren), reduziert den Lernaufwand und kann letztendlich zum Erfolg beitragen.	Speziell in größeren Unternehmen, die eine strikte Trennung zwischen Forschungs- und Anwendungsarbeit besitzen, besteht die Gefahr, daß das Pilotprojekt, in dem erstmals Softwarevermessung angewendet wird, durch die Forschungsmitarbeiter bearbeitet wird, das Management den Transfer auf reale Projekte beschließt und dann die neue Technik – dann allerdings ohne Pilotprojektteilnehmer – auf die anderen Bereiche ausgeweitet wird. Diese sehen sich dann mit einer für sie bis dahin neuen Technik konfrontiert, deren Durchführung und Nutzen sie zumeist nicht verstehen. Ein weiteres Problem dieser häufig als *Metrik-Team* bezeichneten separaten Personengruppe besteht darin, „[...] *daß das Metrik-Team Metrik um der Metrik Willen betreibt und dabei den Blick für die realen Unternehmensziele verliert* " [Rudo94], S. 39. Diese Gefahr hat bereits mehrfach zum Einstellen der Produktmessungsaktivitäten geführt (z.B. innerhalb der australischen Fluggesellschaft *QANTAS* und der Bankgruppe *Westpac*, vgl. [ebd.]).

Werkzeuge

Der Erfolg neuer Methoden hängt häufig stark von der Verfügbarkeit entsprechender Werkzeuge ab. Diesbezüglich können folgende Problemaspekte identifiziert werden (vgl. [Sieg84], [Wall90]):

Risiken der Werkzeugunterstützung zur Einführung neuer Techniken...	**... konkretisiert für die Technik Softwareproduktvermessung**
Die Werkzeuge müssen auch für größere Systeme zuverlässig sein. Darüber hinaus gelten die normalen Anforderungen der Gebrauchstauglichkeit (vgl. ISO 9241 in [Rich95]), d.h. daß auch Aspekte wie Benutzungsfreundlichkeit oder befriedigende Antwortzeiten berücksichtigt werden müssen.	Softwareprodukt-Meßwerkzeuge müssen sowohl auf der Quelltextseite als auch auf der Meßwertseite in der Lage sein, mit großen, praxisrelevanten Datenmengen umzugehen, was spezielle Techniken innerhalb des Werkzeugs erfordert. Diese Anforderung wird von vielen Werkzeugen nicht erfüllt. So wird z.B. in [WTMS95] von einem Experiment berichtet, in dem das RIGI-Werkzeug aufgrund der Größe des zu untersuchenden Systems in allen wesentlichen Teilen überarbeitet werden mußte: „*The sheer amount of code compelled changes to all three components of the environment*" ([ebd.] S. 50). Darüber hinaus werden Meßwerkzeuge häufig als prototypische Shell-Programme vertrieben, was ein benutzungsfreundliches Arbeiten i.d.R. erschwert. So ist die textuelle Ausgabe großer Mengen von Meßwerten der Einführung von Softwarevermessung nicht dienlich, sondern sollte durch eine benutzungsfreundliche Navigation und GUI-gestützte Visualisierung ersetzt werden.

Die Werkzeuge müssen an die spezielle Ablauforganisation des Entwicklungsprozesses angepaßt werden können und eine entsprechende Individualisierung des neuen methodischen Vorgehens unterstützen.	Softwareprodukt-Meßwerkzeuge werden i.d.R. als fertige, kaum anpaßbare Produkte vertrieben. Die Abhängigkeit von Qualität zu den jeweiligen Projekt- und Unternehmenszielen (vgl. Kapitel 2.2) sowie zu den daraus abgeleiteten Qualitätsmerkmalen und Maßen bleiben daher ebenso unberücksichtigt wie die häufige Notwendigkeit der Anpassung verschiedener Parameter der Meßmaschinerie an veränderte Einflußfaktoren. Auch die verschiedenen Prozesse, in denen Maße sinnvoll eingesetzt werden können (vgl. Abschnitt 3.3), werden selten durch Meßwerkzeuge unterstützt, da sich diese lediglich auf die reine Vermessung konzentrieren.

3.6.2 Spezifische Probleme beim Produktmaßeinsatz

In diesem Abschnitt werden für den Produktmaßeinsatz spezifische Probleme beschrieben, die größtenteils auf Erfahrungen von im Rahmen dieser Arbeit durchgeführten Projekten beruhen. Viele dieser spezifischen Probleme haben direkte Auswirkungen auf die allgemeinen Probleme der Technikeinführung; dennoch sollen sie hier noch einmal konzentriert vorgestellt werden, um darauf aufbauend die weitere Zielsetzung dieser Arbeit möglichst konkret beschreiben zu können.

1. *Maß-Selektions-Problem*: Aus der Literatur sind mehr als 1500 verschiedene Softwaremaße bekannt ([Zuse98], S. 10f), deren Selektion für einen speziellen Kontext und für ein angepaßtes Qualitätsmodell sehr schwierig ist; dies wird durch mögliche Korrelationen zwischen den einzelnen Maßen noch verstärkt. Die Zusammenfassung von Softwaremaßen in sogenannte *Metrics-Suites* (z.B. [ChKe94] oder [HaCoNi98]) hilft ebenfalls wenig, da auch hiervon eine Vielzahl mit jeweils eigenen Charakteristika in der Literatur existiert.
2. *Maß-Definitions-Problem*: Die Maße sind selten exakt definiert. So fehlen häufig das zugrundeliegende Produktmodell, eine exakte, auf dem Produktmodell basierende Definition und eine Interpretationshilfe (z.B. welche Werte wünschenswert sind). Das führte in der Vergangenheit zu unterschiedlichsten Interpretationen von Maßen (vgl. Varianten von LCOM in [Hend96], S. 142ff), was wiederum zu nicht vergleichbaren Meßwerten führte. Erste Versuche, Maße streng formal zu definieren (vgl. [BrDaWü97], [BrDaWü99]), sind allerdings immer noch nicht völlig eindeutig (z.B. die exakte Definition, was als *Klasse* verstanden wird; Problemfälle sind z.B. `structs` in C++ oder `interfaces` in Java); sie leiden darüber hinaus i.d.R. zusätzlich an einer schwierigen Vermittelbarkeit und Interpretation der Meßwerte, was zu einem deutlichen Arbeitsmehraufwand führt und eine große Verbreitung des Konzepts der Softwarevermessung behindert.
3. *Wert-Interpretations-Problem*: Maße besitzen häufig kein intuitives Pendant in der realen Welt (vgl. Kapitel 4), was eine Interpretation der Meßwerte und vor allen Dingen das Erkennen von konstruktiven Gegenmaßnahmen erschwert. So wird z.B. in [Müll97] von einem Wartungsmaß berichtet, das sowohl trigonometrische Funktionen, den Logarithmus und die Wurzelfunktion verwendet. Die mittels eines solchen Maßes ermittelte Zahl läßt kaum Rückschlüsse auf die Ursachen für einen bestimmten Meßwert zu und ist damit wertlos.
4. *Messen-Voraussagen-Problem*: Häufig wird Messen mit Voraussagen verwechselt (vgl. [Biem96], S. 85f): Auch wenn Aussagen über zukünftiges Verhalten eines Softwareprodukts wünschenswert sind, so bestimmt das Messen lediglich die graduelle Ausprägung einer Eigenschaft zum Zeitpunkt des Messens. Mögliche Extrapolationen von mehreren Gegenwartsmessungen auf zukünftige Werte bedürfen neben einer statistischen Begründung einer kausalen Erläuterung. Anderenfalls ist die Gefahr der Zufallskorrelationen („*shotgun-correlations*“) gegeben [CoGu93], die allerdings auch heute noch häufig übersehen wird.

5. *Qualitätsmodell-Problem*: Softwaremanager und Softwarenutzer sind primär an der direkten Vermessung externer Merkmale wie z.B. Wartbarkeit oder Zuverlässigkeit interessiert ([FeWhIi95], S. 7f). Die Notwendigkeit, diese sehr allgemeinen Merkmale erst mittels angepaßter Qualitätsmerkmale (vgl. Kapitel 2) auf die Ebene meßbarer Eigenschaften zu verfeinern und die damit häufig implizit gemachte Reduktion der Anforderungen auf tatsächlich meßbare Eigenschaften ist häufig unbekannt. Daher wird in der Praxis oftmals mit stark vereinfachten Maßen gearbeitet (z.B. für Qualität), deren Interpretation häufig zu Problemen führt (vgl. [ebd.]).
6. *Wert-Rückprojektions-Problem*: Die Abstraktion von Quelltext auf Softwaremaße ist so stark, daß die Rückprojektion eines Wertes auf den ihn begründenden Softwareteil sehr schwierig ist. Während quelltextnähere Abstraktionen – wie z.B. Klassendiagramme – ein einfacheres Abbilden der dortigen Strukturen auf den Quelltext ermöglichen, ist dies für bloße numerische Werte äußerst schwierig. Dies zeigt sich besonders in Darstellungen, bei denen einzelne Entitäten mit besonders kritischen Meßwerten identifiziert werden sollen (vgl. Transformationstechniken in Abschnitt 3.4). Die bloße Präsentation überlanger Listen erschwert diese Rückprojektion stark, so daß häufig nur noch mit den Zahlen gearbeitet wird, ohne jeweils die (Zwischen-) Ergebnisse in den eigentlichen Problembereich zurück zu übertragen.
7. *Multiwert-Problem*: Ein angepaßtes Qualitätsmodell umfaßt i.d.R. viele unterschiedliche Maße für ein und dieselbe vermessene Entität. Viele Interpretationen bzgl. einer vermessenen Entität basieren nicht auf der Einzelbetrachtung eines Wertes eines Softwaremaßes, sondern auf der Kombination bzw. Korrelation der Meßwerte verschiedener Maße. Die bisherigen Visualisierungen für diese *Multimetriken* ([ErLe96]) wie z.B. Kiviatdiagramme sind für große Systeme ungeeignet, da sie jeweils nur die Meßwerte einer Entität ohne weiteren Kontext darstellen.
8. *Meßtheorie-Anwendungs-Problem*: Robert Glass formuliert dieses Problem wie folgt: „*Theory and Practice are traveling very different roads on the topic of software metrics*“ ([Glas96], S.11). Während die Praktiker entweder gar nicht messen oder mit sehr fragwürdigen, nicht validierbaren Maßen arbeiten, erarbeiten die Theoretiker häufig sogenannte Anforderungskataloge an Softwaremaße (z.B. die Weyuker-Eigenschaften [Weyu88] oder die Zuse-Anforderungen, [Zuse98], Kapitel 7), die untereinander nicht konsistent, für den Entwickler häufig nicht intuitiv und für eine praktische Anwendung in Form eines Meßwerkzeugs zu viele, nicht automatisch extrahierbare Informationen benötigen. Ein wesentlicher Grund dafür ist die häufig zu hohe Anforderung an das empirische Relationensystem und der damit verbundene Mindestskalentyp der Rationalskala. (vgl. Kapitel 4).
9. *Kopplungs-Kohäsions-Problem*: Bei den Strukturmodellen wird i.d.R. deutlich zwischen den Kopplungsmodellen und den Kohäsionsmodellen unterschieden. Beide werden häufig weiter verfeinert (für die Kopplung z.B. in Benutzungskopplungen und Vererbungskopplungen). Diese scheinbar orthogonal zueinander stehenden Modelle werden jeweils einzeln betrachtet, validiert und angewendet. Eine gemeinsame, diese Modelle subsumierende Sicht, die es ermöglicht, sowohl Kopplung als auch Kohäsion gleichermaßen zu modellieren, existiert nicht. Dadurch werden ebenfalls Abhängigkeiten zwischen den verschiedenen Modellen nicht betrachtet (z.B. kann Kohäsion durch Kopplung begründet sein, vgl. Kapitel 7).
10. *Vererbungs-Meßwerte-Problem*: Speziell für objektorientierte Systeme birgt die ausschließlich separate Betrachtung von Vererbung mittels Vererbungsmaßen eine große Gefahr bei der Meßwertanalyse klassischer Größen-, Kopplungs- und Kohäsionsmaße, da jene bei der Betrachtung von aus der Vererbungsstruktur separierten Klassen nur Teilausschnitte erfassen können (vgl. [SiBe00]).
11. *Relevanz-Problem*: Der gesamte Bereich der Softwarevermessung zur Qualitätsverbesserung ist lediglich für größere Softwaresysteme relevant, da bei kleineren Systemen die direkte

Möglichkeit einer manuellen Inspektion oder gar der Neuimplementierung den Mehraufwand einer Einführung von Softwarevermessung i.d.R. nicht rechtfertigt. Auf der anderen Seite existieren meßbasierte Softwareproduktanalysen, die entweder nur semiautomatisch funktionieren (vgl. z.B. Abschnitt 3.2.3) oder die aufgrund äußerst rechen- und datenintensiver Algorithmen nicht beliebig skalierbar sind (wie z.B. Maße aufbauend auf einer um die Zeigerarithmetik angereicherten *Points-To-Analyse* auf einer *Slicing*-Darstellung eines Systems, vgl. [KrRo99], [StSn00]). In beiden Fällen ist deren Einsatz in Softwareprodukt-Meßwerkzeugen für relevante Systemgrößen praktisch nicht möglich, was zu einem deutlich eingeschränkten Anwendungsgebiet für diese Maße führt.

3.6.3 Weitere Zielsetzung dieser Arbeit

Das Ziel dieser Arbeit ist, den im vorigen Abschnitt vorgestellten spezifischen Problemen beim Einsatz von Softwareproduktmaßen mit neuen Konzepten erfolgreich zu begegnen und deren Praxisrelevanz praktisch zu belegen. Für diesen Zweck werden folgende, teilweise aufeinander aufbauende Teilaufgaben in dieser Arbeit behandelt:

In Kapitel 2 wurde bereits das Qualitätsmodell-Problem diskutiert, dem in Kapitel 3 der bisherige Stand der Wissenschaft (inkl. dem Aufzeigen des Relevanz-Problems entlang verschiedener Aspekte) gegenübergestellt wird. Das Ergebnis von Qualitätsmodellen in Form der möglichst stark verfeinerten Qualitätsmerkmale wird in Kapitel 4 in den Kontext der Meßtheorie eingebettet. Deren formaler Apparat hilft, den wichtigen Zusammenhang zwischen den Qualitätsmerkmalen und den sie messenden Maßen explizit zu machen und damit das Wert-Interpretations-Problem und das Messen-Voraussagen-Problem zu lösen. Beide Probleme werden noch einmal besonders deutlich bei der Rückübertragung des formalen Apparates auf den praktischen Prozeß des Messens, die hilft, dieses tagtäglich angewendete Konzept besser zu verstehen bzw. die Anforderungen an das Messen explizit machen zu können. Gleichzeitig wird dort das Meßtheorie-Anwendungs-Problem offensichtlich. Um letzteres zu lösen, werden minimale Anforderungen an für Qualitätssicherung einsetzbare Skalen erarbeitet, die sowohl den Ansprüchen der Meßtheorie als auch denen der Praxistauglichkeit genügen; diese Anforderungen unterstützen sowohl die meßtheoretisch-konforme Erstellung neuer Skalen als auch eine Überprüfung existierender Skalen (bzw. deren Maße) und münden in ein „Vorgehensmodell zur Vermessung von Softwareprodukten mittels intern valider Maße“. Dessen konsequente Anwendung löst das Maß-Definitions-Problem und reduziert, durch die häufig möglich werdende Reduktion der Aussagekraft vieler existierender Maße, das Maß-Selektions-Problem.

Letzteres wird weiter durch ein in Kapitel 5 vorgestelltes Graphenmodell bisheriger Maße reduziert, da es grundsätzliche Gemeinsamkeiten und Unterschiede von verschiedenen, bereits in diesem Kapitel erarbeiteten Produktmaßklassen hervorhebt. Damit wird ebenfalls eine neue Sicht auf das Wert-Rückprojektions-Problem deutlich, das anschließend die Erstellung einer neuen Art von Maßen motiviert. Diese durch ein in Kapitel 6 vorgestelltes generisches Distanzmaß erzeugbaren echten Metriken reduzieren alle im letzten Abschnitt aufgeführten spezifischen Probleme, da sie

- eine echte Erweiterung bisheriger Maße darstellen und somit nicht nur eine weitere Maßart darstellen, sondern mittels der durch sie *zusätzlich* gewonnenen Informationen helfen, die spezifischen Meßprobleme bei gleichzeitiger Anwendung bisheriger klassischer Maße zu lösen,
- auf einem dem Menschen sehr zugänglichen Konzept der Ähnlichkeit beruhen, das, nach entsprechender, der Meßtheorie konformen Instanziierung, die durch zwei wohldefinierte, auf dem „Vorgehensmodell zur Vermessung von Softwareprodukten mittels intern valider Maße“ aufbauenden Prozesse unterstützt wird, sinnvolle Ergebnisse liefert,
- die einfache Anpaßbarkeit an individuelle Anforderungen unterstützen, und da sie

- aufgrund ihres Metrik-Charakters für den Bereich der Softwarevermessung neue Techniken anzuwenden erlauben, die auch für große Systeme noch effizient die gewünschten Ergebnisse erkennen lassen.

Diese Techniken, namentlich die Visualisierung und die Clusteranalyse, erlauben eine deutliche Reduktion des Wert-Rückprojektions-Problems und des Multiwert-Problems. Erste Instanziierungen dieses generischen Distanzmaßes werden in Kapitel 7 zur Nachbildung einiger bekannter Kohäsionsmaße verwendet, um noch einmal das Maß-Selektions und das Maß-Definitions-Problem zu reduzieren und gleichzeitig Lösungen zum Kopplungs-Kohäsions-Problem vorzustellen. Diese werden in Kapitel 8 spezifisch für objektorientierte Systeme erfolgreich angewendet; das für objektorientierte Systeme typische Vererbungs-Meßwerte-Problem wird dort zuvor mittels einer neuen Sicht auf Vererbungsstukturen gelöst. Deren Relevanz wird sowohl für klassische Maße als auch für die Metriken theoretisch und praktisch demonstriert.
In Kapitel 9 wird anhand einiger, im Rahmen dieser Arbeit durchgeführten Projekte das Lösen der spezifischen Probleme beim Produktmaßeinsatz mit den in dieser Arbeit vorgestellten Konzepten praktisch demonstriert. Damit wird abschließend ebenfalls eine wesentliche, in der Literatur aber unzureichend vorgenommene Anforderung der Meßtheorie – die externe Validierung von Maßen – durchgeführt. Die ebenfalls in Kapitel 9 vorgestellte, vollständige Meßwerkzeugumgebung (inkl. des in diesem Kapitel bereits vorgestellten Produktmeßwerkzeugs Crocodile) demonstriert eine Lösung des Relevanz-Problems für objektorientierte Systeme mit mehreren 1000 Klassen.

3.7 Zusammenfassung

Softwaremaße sind ein Schlüsselelement des Software-Engineering. Sie sind in der Lage, in wichtigen Feedback-Loops die Aufgabe des Überprüfens zwischen dem Ist- und Sollzustand zu unterstützen und gegebenenfalls rechtzeitig Gegenmaßnahmen zu motivieren. Dies gilt für die Bereiche Softwareprodukte, Softwareprozesse und Softwareressourcen, die mittels Produktmaßen, Prozeßmaßen und Ressourcenmaßen vermessen werden können. Der Schwerpunkt dieser Arbeit liegt auf den Produktmaßen, da die Produkte als das Ergebnis von Ressourcen benötigenden Prozessen letztendlich den wirtschaftlichen Erfolg eines Unternehmens bestimmen. Die vorgenommene Einteilung von Produktmaßen in Größen-, Kopplungs- und Kohäsionsmaße entspricht dabei der Art der Erstellung des Produktmodells, d.h. der Art der Abstraktion vom zugrundeliegenden Produkt.
Die verschiedenen Prozesse innerhalb des Software-Engineering, in denen Softwaremaße sinnvoll eingesetzt werden können, sind die Produktqualitätsbeurteilung, die Review-Vorbereitung, die Programmüberarbeitung und die Trendanalyse. In allen Fällen spielt die effiziente Erzeugung der für Softwareprodukte ermittelten Meßwerte eine wesentliche Rolle, so daß die Notwendigkeit der Werkzeugunterstützung für Softwaremaße als inhärent bezeichnet werden kann. Speziell für große Systeme genügt allerdings die bloße Erzeugung von Meßwerten nicht mehr, da die Interpretation der Meßwerte aufgrund deren Zahlenmenge sehr aufwendig ist. Erst die Anwendung verschiedener Transformationstechniken, die klassifiziert werden in Statistik, Wertefilter, Komponentenfilterung und Diagramme, ermöglicht z.B. die Softwarevisualisierung, die es dem Benutzer erlaubt, die Ausgangsinformationen bestmöglich zu verstehen.
Die verschiedenen Charakteristika eines Produktmeßwerkzeugs und deren praxisrelevante Ausprägungen werden u.a. anhand des Werkzeugs Crocodile demonstriert: Es erlaubt aus der dem Entwickler bekannten Entwicklungsumgebung heraus das angepaßte, statistisch aufbereitete, filterbare und archivierbare Vermessen von (unvollständigen) C++- und JAVA-Quellen und unterstützt dessen Interpretation durch verschiedene Visualisierungen und Rückprojektionen auf das zugrundeliegende System.
Die dennoch bei der Anwendung klassischer Softwaremaße auftretenden Probleme entsprechen zu großen Teilen den grundsätzlichen Problemen der Neueinführung von Techniken oder Werkzeugen innerhalb des Software-Engineering. Die dort aufgezeigten Problembereiche Vorgehen, Management, Promotoren, Betroffene, Ausbildung und Unterstützung, Pilotprojekte und Werkzeuge dienen daher als gute Strukturierung der Probleme, die bis heute den verbreiteten Einsatz von Produktmaßen innerhalb der Softwarebearbeitung behindert haben. Viele spezifische Probleme des Produktmaßeinsatzes haben folglich direkte Auswirkungen auf die allgemeinen Probleme, werden allerdings dennoch für eine genaue Zielsetzung der weiteren Arbeit explizit aufgezählt und lassen eine Klassifizierung in das Maß-Selektions-Problem, das Maß-Definitions-Problem, das Wert-Interpretations-Problem, das Messen-Voraussagen-Problem, das Qualitätsmodell-Problem, das Wert-Rückprojektions-Problem, das Multiwert-Problem, das Meßtheorie-Anwendungs-Problem, das Kopplungs-Kohäsions-Problem, das Vererbungs-Meßwerte-Problem und in das Relevanz-Problem zu.

„Theory and practise
are traveling very different roads
on the topic of software metrics"
([Glas96], S. 13)

4 Meßtheorie

Viele der im vorigen Kapitel aufgezeigten Probleme des effizienten Einsatzes von Softwareproduktvermessung für die Qualitätssicherung (vgl. Kapitel 3.6) liegen in der großen Erfahrung beim täglichen Umgang mit Maßen begründet: Sie sind i.d.R. einfach anzuwenden (z.B. Gewicht), erlauben auf effiziente Art und Weise den schnellen Vergleich mit einer Vielzahl anderer gemessener Dinge (z.B. wer ist der schwerste) und lassen sich darüber hinaus auch häufig zu verschiedenen kombinierten Maßen – sogenannte *hybride Maß*e, die sich aus der Kombination mehrerer Maße ergeben – zusammensetzen (z.B. wer ist der schnellste als das Verhältnis von Weg zu Zeit). Da viele der Maße allerdings tradiert sind und daher häufig lediglich pragmatisch angewendet werden, sind die genauen Parameter, unter denen Messungen sinnvoll vorgenommen werden können, unbekannt (vgl. [Zuse98], Kapitel 2.4.2). Diese Kombination aus auf Erfahrung beruhender Erwartungshaltung gegenüber Maßen und den nicht explizit gemachten Anforderungen an Maße birgt eine große Gefahr beim Transfer des Meßkonzepts auf den Bereich der Softwarevermessung: gerade hinsichtlich des noch relativ jungen Alters des Wissenschaftsbereichs Software-Engineering und des damit begründeten unvollständigen Wissens im Bereich großer Softwaresysteme erscheinen Maße oftmals als Universallösung softwaretechnischer Probleme. Häufig sind Enttäuschungen aufgrund dieser deutlich zu hoch gesteckten Anforderungen an Maße das Ergebnis.
Die *Meßtheorie*[5] als Modell des Vermessens versucht nun, eine Theorie nachzuliefern, anhand derer folgende Fragen beantwortet werden können (vgl. [ebd.], Kapitel 4.2):

- Welche Voraussetzungen müssen für das Messen erfüllt sein?
- Was ist meßbar, was ist nicht meßbar?
- Was mißt ein Maß?
- Wie sind die Meßwerte eines Maßes zu interpretieren?
- Wie können Maße validiert werden?

Während Softwarevermessung bereits ab Mitte der fünfziger Jahre angewendet wurde (z.B. das Maß LOC für eine Größenanalyse des ersten FORTRAN-Compilers, [Back56]; oder der erste Versuch, Komplexität von Software mittels auf 7 Kategorien verteilten 57 Eigenschaften zu vermessen, [RuHa68]), begann die Anwendung der Meßtheorie im Bereich der Softwarevermessung erst Mitte der achtziger Jahre (z.B. [ZuBo85]). Sie stützt sich dabei auf allgemeine, die Meßtheorie behandelnde Arbeiten von Roberts [Robe79] und diese wiederum auf die „*Meilensteinarbeit*" von Krantz [Kran et al.71] (vgl. [SaEh92], S. 5)[6].

[5] Es wird deutlich zwischen *Meßtheorie* (engl. *measurement theory*) und *Maßtheorie* (engl. *measure theory*) unterschieden. Während die Maßtheorie ihren Schwerpunkt auf die Integralberechnung in metrischen Räumen setzt, beschäftigt sich die Meßtheorie mit Relationensystemen (Abschnitt 4.1.3) und Abbildungen (Abschnitt 4.2) dazwischen (vgl. [Zuse98], S. 356).

[6] Für eine detaillierte historische Genese der Meßtheorie von den Griechen, über Descartes bis hin zu Helmholtz, Hölder und eben Krantz siehe [SaEh92].

Mittlerweile besitzt jedes Buch, das sich dem Thema Softwarevermessung widmet, einen eigenen Abschnitt zum Thema Meßtheorie. Allerdings werden die dort beschriebenen Anforderungen, die eingeführten Begriffe und die erarbeiteten Ergebnisse nur selten für den praktischen Teil der Softwarevermessung verwendet. Die von Glass festgestellte Diskrepanz zwischen Theorie und Praxis (s.o.) trifft daher nach wie vor zu.
Dieser Abschnitt versucht nun, die Meßtheorie für den Bereich der Softwarevermessung derartig aufzubereiten, daß sie als Anforderungskatalog, Nomenklatur und Durchführungsrichtlinie verwendet werden kann und in den folgenden Kapiteln auch verwendet wird. Sie basiert dabei wie bei Zuse [Zuse98], Hendersson-Sellers [Hend96] und Fenton [FePf96] auf dem Meßtheorie-Klassiker von Krantz [Kran et al.71]. Zusätzlich wird die Arbeit von Pfanzagl herangezogen [Pfan71], die den Bereich mit präzisen Definitionen aufzeigt und mit vielen anschaulichen Beispielen anreichert.
Die in diesem Abschnitt abgeleiteten Anforderungen an ein erfolgreiches Messen sind deutlich weniger restriktiv als z.B. bei Zuse, der grundsätzlich die Existenz einer *extensiven Struktur*, d.h. einer Verknüpfungsoperation zwischen den zu vermessenden Elemente voraussetzt (vgl. Abschnitt 4.1.2), und sie sind ausschließlich auf Anforderungen an das Verständnis der zu vermessenden Software ausgerichtet (vgl. Meßtheorie-Anwendungs-Problem in Kapitel 3.6.2).
In diesem Abschnitt wird sorgfältig zwischen den beiden Konzepten *Messung* und *Vorhersage* unterschieden (vgl. Messen-Voraussagen-Problem in Kapitel 3.6.2): Während das erste die gegenwärtige Ausprägung eines betrachteten Merkmals bestimmt, versucht das zweite – aufbauend auf den erhaltenen Meßwerten – Aussagen über das zukünftige Verhalten einer Entität zu machen. Diese Unterscheidung wurde bereits von der sogenannten *Grubstake-Gruppe*[7] angemahnt: „*It arises out of a basic (and poorly articulated) misconception that a software measure must always be part of a prediction system.*“ ([Baker et al90a], S. 279). Diese Arbeit beschränkt sich auf die Untersuchung des Konzepts *Messung*, da selbst dieses noch viele Fragen aufwirft und es unzureichende Erfahrung damit gibt. Das Konzept der *Vorhersage*, das auf einer solchen Messung basiert und darauf aufbauend Aussagen über das zukünftige Verhalten machen soll, ist für den Bereich des Software-Engineering erst ein längerfristiges Ziel. Zuvor sollte der Bereich Messung sowohl verstanden und etabliert sein, als auch eine wissenschaftliche Herangehensweise ermöglichen.
Die derzeit noch präsente Schwierigkeit, Vorhersagesysteme detailliert herzuleiten, steht allerdings diametral zur Forderung, mittels Maßen Probleme vorherzusagen, die in der Zukunft auftreten können (z.B. hohe Wartungskosten). Diese Diskrepanz kann durch die Definition eines angepaßten Qualitätsmodells gelöst werden (vgl. Kapitel 2): Die in die Zukunft gerichtete Verfeinerung des Qualitätsbegriffs (z.B. durch Merkmale wie Wartbarkeit, Portierbarkeit oder Wiederverwendbarkeit) ist hierbei bis auf die Ebene derjenigen Maße fortzuführen, deren gegenwärtige Meßwerte erfahrungsgemäß Aussagen über das Qualitätsmerkmal und damit auf Eigenschaften in der Zukunft erlauben. Diese Einschätzung hängt allerdings von einer Vielzahl von Parametern ab (vgl. Kapitel 2.2) und wird nicht mehr als Messung, sondern als Schätzung betrachtet. In einem solchen Fall unterstützt der Wert einer Messung, der die aktuelle Ausprägung eines Merkmals bestimmt, die Einschätzung über zukünftiges Verhalten, er mißt letzteres aber nicht. Der vorsichtige, mündige Umgang mit Meßwerten in allen im 3. Kapitel vorgestellten Prozessen berücksichtigt bereits diese *Messung-Schätzung-Unterscheidung*.

[7] Die *Grubstake-Gruppe* setzt sich aus den Wissenschaftlern Albert L. Baker, James M. Bieman, Norman Fenton, David A. Gustafson, Austin Melton und Robin Whitty zusammen [Baker et al.90a] und versucht, die Defizite der Meßanwendung innerhalb der Software-Technik durch grundlegende, allgemeingültige Arbeiten zu kompensieren.

Das Schlüsselelement der *Relationensysteme* als Ausdruck des gegenwärtigen Verständnisses von Systemen wird in Abschnitt 4.1. daher detailliert und separat behandelt. Bereits hier werden, aufbauend auf Erfahrungen von im Rahmen dieser Arbeit durchgeführten Projekte, Kriterien aufgezeigt, die in der Praxis der Softwareproduktvermessung auftretende Probleme der Erstellung von Relationensystemen zu vermeiden helfen. Darauf aufbauend wird die Erstellung einer Skala mit den meßtheoretisch relevanten Problemen *Repräsentationsproblem* (engl.: *representational problem*), *Eindeutigkeitsproblem* (engl.: *uniqueness problem*) und *Aussagekraftsproblem* (engl.: *problem of meaningfulness*) in Abschnitt 4.2. angesprochen. Abschnitt 4.3. behandelt das Problem und die Notwendigkeit der internen und externen Maßvalidierung. Um die aufgezeigte Diskrepanz zwischen Theorie und Praxis im Bereich der Softwareproduktvermessung zu beseitigen (vgl. Meßtheorie-Anwendungs-Problem in Kapitel 3.6.2), wird, unter Berücksichtigung der aufbereiteten Ergebnisse der Meßtheorie und den Anforderungen nach Validierung, in Abschnitt 4.5. ein neuer, konstruktiver und in eigenen praktischen Arbeiten erfolgreich eingesetzter Prozeß zur meßtheoretisch konformen und zielorientierten Skalendefinition abgeleitet.

4.1 Merkmale und Relationen

Das Hauptaugenmerk jedes Meßvorgangs liegt in der Betrachtung eines *Merkmals* (engl.: *property*) einer Entität ([Pfan71], S. 15):

> Ein *Merkmal* ist eine charakteristische, kennzeichnende Eigenschaft, durch die eine Entität von anderen unterschieden ist.

Definition 15: Merkmal (entsprechend *Meyer*-Lexikon)

Jede Entität besitzt Merkmale in unterschiedlichen Ausprägungen. Diese Merkmale können u.a. zur Anwendung des *principium individuationis* verwendet werden, d.h. für beliebige Entitäten e_1 und e_2 gilt, daß wenn e_1 exakt dieselben Merkmale in denselben Ausprägungen wie e_2 besitzt, dann sind e_1 und e_2 identisch ([Bung77], S.74).
Das Ziel von Messungen ist aber häufig, bestimmte *Eigenschaften* (engl.: *attribute*) von Entitäten zu bestimmen:

> Eine *Eigenschaft* ist die einer Entität zukommende Bedeutung.

Definition 16: Eigenschaft (entsprechend *Meyer*-Lexikon)

Während in der Literatur beide Begriffe häufig synonym verwendet werden (z.B. [Pfan71], S. 15; [FePf96], S. 5), wird hier deutlich zwischen ihnen unterschieden, da das Messen von Merkmalen aufgrund deren Objektivität – schließlich sind Merkmale auch ohne menschliches Zutun vorhanden – deutlich einfacher ist als das Messen von Eigenschaften, die vom jeweiligen Kontext abhängen, in dem sie bestimmt werden sollen. So läßt sich beispielsweise das Merkmal Höhe eines Gebäudes leichter bestimmen als dessen Eigenschaft Schönheit, die speziell für den weiteren lokalen Kontext eines Gebäudes (z.B. als Teil einer Häuserfront) durchaus eine diesem zukommende Bedeutung darstellt. Eigenschaften einer Entität können erst dann gemessen werden, wenn bzgl. ihrer Bedeutung für eine Entität Konsens herrscht. Dies ist z.B. für die Eigenschaften „gefühlte Temperatur“ (als Kombination mehrerer Merkmale wie Luftfeuchtigkeit, Temperatur, Luftdruck etc.) und viele Sicherheitsnormen im Ingenieurwesen erfüllt. Das Nichtbeachten des Konsens als Voraussetzung führte in der Vergangenheit z.B. innerhalb der Psychologie zu kuriosen Messungen, wie z.B. die Messung der Handschriftqualität, der Beurteilung von Sonnenuntergängen oder die graduelle Bestimmung der Einstellung zum Kommunismus ([SaEh92], S. 3).
Der notwendige Konsens selbst kann auf allgemeiner Einsicht oder auf einer allgemeinen Regel beruhen. Da im Softwarebereich eine allgemeine Einsicht über Eigenschaften noch nicht etabliert ist, besteht die Möglichkeit, Regeln, soweit sie nicht bisheriger Einsicht widersprechen, zu

definieren. So setzt McCabe für einen Programmteil die Eigenschaft „komplex“, für die per se keine einheitliche Vorstellung existiert, gleich mit dem Besitz von mehr als 10 linear unabhängigen Programmpfaden (vgl. [MCCA82]). Diese Möglichkeit, Eigenschaften, die von sich aus keine natürliche Konsensfähigkeit mitbringen, zu definieren, um sie damit meßbar zu machen, kann Teil des Qualitätsmodells sein (vgl. Kapitel 2): Die Verfeinerung kann mit den oben vorgestellten Definitionen als die Zuordnung von Merkmalen zu Eigenschaften aufgefaßt werden, um letztere bestimmen zu können. Wie bereits dort beschrieben, sind diese Zuordnungen immer diskutabel und müssen dem jeweiligen Kontext angepaßt werden. Im Folgenden wird das Messen auf das Messen von Merkmalen beschränkt. Erst deren Etablierung innerhalb des Software-Engineering wird in Zukunft einen auf Einsicht basierenden Konsens über Eigenschaften ermöglichen.
Die Beschränkung der Betrachtung auf Merkmale einer Entität garantiert allerdings immer noch nicht deren direkte Meßbarkeit (vgl. Kapitel 3.1): So läßt sich z.B. das Merkmal Geschwindigkeit nur indirekt durch die gleichzeitige Betrachtung einer Zeitmessung und einer Wegmessung bestimmen (vgl. [SaEh92], S. 2).
Eine wichtige Voraussetzung jedes Messens ist also die bewußte Konzentration auf bestimmte Merkmale der zu vermessenden Entität. Um irrelevante Merkmale auszublenden, wird die Entität entsprechend der gewählten Konzentration vereinfacht: Ein Modell der Entität entsteht (vgl. Produktmodell in Kapitel 3.2). Im folgenden Kapitel wird die Produktmodellerstellung für Softwareprodukte vorgestellt, bevor darauf aufbauend Relationen auf diesen Modellen hergeleitet werden und damit die Definition von Relationensystemen möglich ist.

4.1.1 Produktmodelle

Die Erstellung eines Modells geschieht innerhalb des tagtäglichen Messens häufig implizit. So wird beispielsweise bei der Größenbestimmung von Personen von allen für die Größenbestimmung irrelevanten Merkmalen abstrahiert. Dies umfaßt Merkmale wie Haarfarbe, Körperumfang oder Geschlecht. Für die Messung wird also implizit ein sehr einfaches Modell erstellt, wie es in Abbildung 12 dargestellt ist.

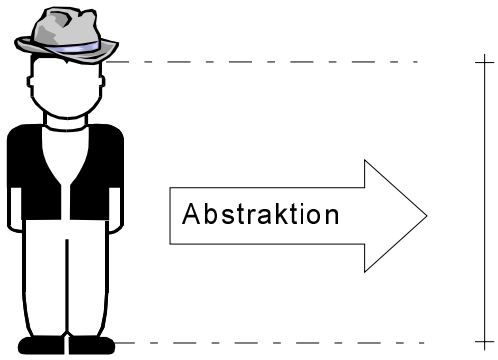


Abbildung 12: Implizite Abstraktion bei Größenmessung von Personen

Während bei allgemein üblichen Messungen die Modelle häufig klar sind, ist dies für den Bereich der Softwarevermessung nur selten der Fall. Bevor der Bereich der Produktmodelle von Software detailliert vorgestellt wird, sollen im folgenden anhand des Größenbeispiels typische, im Rahmen dieser Arbeit identifizierte Anforderungen an die Modellierung formuliert werden (vgl. [SiRuLe00]):

- *Vollständigkeit*: Für jedes Merkmal der Entität muß definiert sein, welche Rolle es in der Abstraktion spielt. Bezogen auf das Beispiel umfaßt dies z.B. die Fragen, ob die Schuhhöhe und eventuelle Kopfbedeckungen ebenfalls betrachtet werden.
- *Eindeutigkeit*: Jedes betrachtete und nicht betrachtete Merkmal muß eindeutig spezifiziert sein und darf keine Mißverständnisse erlauben. Bezogen auf das Beispiel umfaßt dies z.B. die Fragen, was unter einer Kopfbedeckung zu verstehen ist und wie z.B. Beinprothesen berücksichtigt werden.
- *Konsistenz*: Die Abstraktion muß konsistent für alle betrachteten Entitäten vorgenommen werden, d.h. das Vorkommen spezieller Merkmale sollte konsistent für alle Entitäten gleich behandelt werden. Bezogen auf das Beispiel umfaßt das z.B. Überlegungen, ob bei Männern

die relativ geringe Schuhhöhe mit modelliert werden darf, wohingegen bei Frauen dies aufgrund der eventuell deutlich höheren Schuhhöhe nicht gemacht wird. Solche Inkonsistenzen sind bei der Abstraktion zu vermeiden.

- *Verständlichkeit*: Die Abstraktion muß verständlich formuliert sein, aber dennoch vollständig, eindeutig und konsistent. Hier ist ein Kompromiß notwendig zwischen formalen Beschreibungstechniken, die häufig die ersten drei Bedingungen erfüllen, aber nicht immer leicht verständlich sind, und umgangssprachlichen Beschreibungstechniken, die häufig zwar einfach verständlich, aber dafür nicht immer vollständig, eindeutig und konsistent sind.

Das aus einer Entität gebildete Modell spielt eine fundamentale Rolle für das Messen: Aufgrund der vorgenommenen, durch die Merkmalsauswahl motivierten Abstraktion und der Konzentration darauf können direkte Aussagen nur noch auf dem Modell vorgenommen werden. Die Qualität der Rückübertragung der Ergebnisse innerhalb der Modelle auf die ihnen zugrundeliegenden Entitäten hängt von der Qualität der Abstraktion ab[8]. So kann eine inkonsistente Modellierung für die Größenbestimmung von Personen, bei der z.B. die Berücksichtigung eventueller Kopfbedeckungen unterschiedlich gehandhabt wird, zu einem Modell führen, aus dem Aussagen nicht mehr sinnvoll auf die eigentlichen Personen zurück übertragen werden können.
Die Wichtigkeit der Abstraktion als Vorbedingung von Softwarevermessung wird erstmals als expliziter Schritt bei der Herleitung von Softwaremaßen von der Grubstake-Gruppe (s.o.) detailliert beschrieben ([Baker et al.90a], [Baker et al.90b]). Dort wird eine Funktion *abs* definiert, die alle betrachteten Entitäten auf ihre Abstraktionen abbildet.
Im Rahmen der praktischen Erfahrung beim Vermessen von Quelltext mit Crocodile (vgl. Kapitel 3.5) wurden alle Produktabstraktionen unabhängig ihrer Klassifizierung mittels drei separater Schritte explizit definiert, um alle oben genannten Anforderungen bestmöglich zu erfüllen [SiRuLe00]:

1. *Umgangssprachliche Definition*: Hier wird die Abstraktion umgangssprachlich vorgestellt. Dabei wird besonderen Wert auf eine möglichst hohe sprachliche Präzision gelegt, um die vier Anforderungen bestmöglich zu erfüllen.
   Folgendes Beispiel beschreibt z.B. eine Abstraktion für C++-Systeme, auf der anschließend für eine Klasse das Merkmal „öffentliche Attribute“ als Indiz der Eigenschaft Datenkapselung vermessen werden kann:

   > *Diese Abstraktion berücksichtigt nur Klassen innerhalb des Quelltextes und ignoriert jegliche Beziehungen zwischen ihnen. Als Klasse innerhalb des Quelltextes wird jede Entität verstanden, die mit den Worten „`class <identifier>{`“ eingeleitet wird. Der Typ der Klasse (abstrakt oder konkret) wird dabei ignoriert. Eingebettete Klassen werden als separate Klassen betrachtet, d.h. die Einbettung wird ebenfalls ignoriert. Innerhalb jeder Klasse werden lediglich Attribute mit der Sichtbarkeit `public` berücksichtigt. Weder der Typ des Attributs (Basistyp oder komplexer Typ) noch die Art der Bindung (Klassenattribut oder Objektattribut) sind relevant. Felder, Strukturen, Objekte oder Pointer werden jeweils als ein Attribut betrachtet. Geerbte Attribute werden nicht berücksichtigt.*

2. *Visualisierung*: In einem zweiten Schritt wird versucht, die Abstraktion möglichst aussagekräftig zu visualisieren. Es geht hierbei nicht um das Hinzufügen neuer Informationen, wie die Abstraktion vorgenommen werden soll, sondern lediglich um eine zusätzliche Hilfestellung. Aufgrund ihrer Kompaktheit kann sie später als Grundlage zur Herleitung von Relationen verwendet werden (s.u.).

[8] Unter Qualität wird hier die Einhaltung der vier oben genannten Anforderungen an eine Abstraktion verstanden.

Für das Beispiel der Merkmalsbestimmung „öffentliche Attribute“ kann eine solche Visualisierung wie in Abbildung 13 dargestellt werden. Auf einen Blick ist ersichtlich, daß Methoden, Assoziationen, Vererbungen und nicht-öffentliche Attribute ignoriert werden und die Unterscheidbarkeit der Klassen nur noch aufgrund ihrer Namen und der betrachteten Merkmale vorgenommen werden kann.

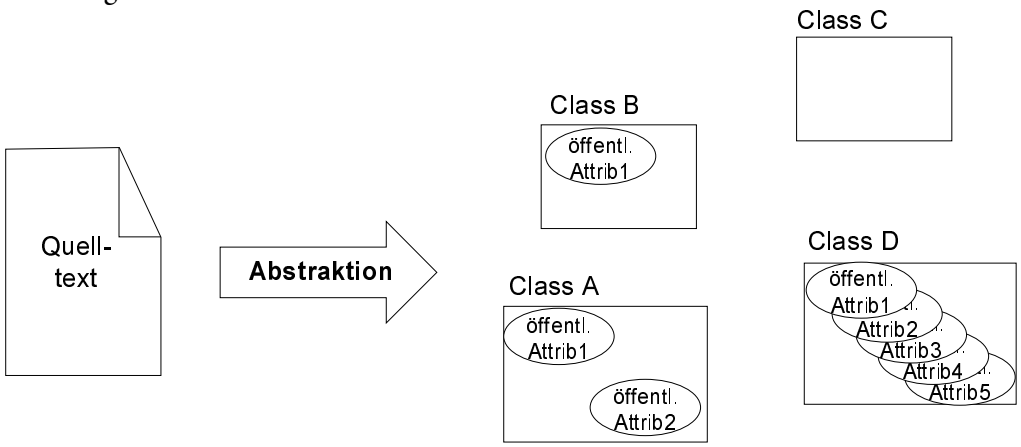


Abbildung 13: Abstraktion für die Betrachtung des Merkmals „öffentliche Attribute“

3. *Beispielabstraktion*: Für die praktische Durchführung der Abstraktion wird in diesem Schritt ein nicht-trivialer Quelltextteil exemplarisch in die Abstraktion überführt. Er sollte die Anwendung aller in der umgangssprachlichen Definition aufgeführten Regeln praktisch demonstrieren.

Die verschiedenen Abstraktionsfunktionen können entsprechend der in Kapitel 3.2. vorgenommenen Klassifizierung in Größen- und Strukturmodelle klassifiziert werden, wobei letztere weiter in Kopplungs- und Kohäsionsmodelle verfeinert werden können. Folglich können folgende Abstraktionsverfahren unterschieden werden:

- *Größenabstraktionen*: Hierbei werden spezielle Systemteile als nicht zusammenhängender Inhalt eines Containers aufgefaßt und dieser bzgl. seiner Inhaltsmenge untersucht. Dies ergibt für einige typische, in Kapitel 3.2.1 aufgezeigten Maße folgende dahinter liegende Abstraktionsfunktionen (hierbei wird die maximal abstrahierende Funktion beschrieben, die für die spätere Vermessung gerade noch genügend Informationen besitzt):
  - *LOC*: Hierbei wird jede im Quelltext vorkommende Programmzeile zu einem Punkt oder Knoten abstrahiert und zu einem für jede Funktion existierenden Container hinzugefügt. In der Abstraktion sind keine Ordnung zwischen den Punkten, keine weitergehenden Informationen wie z.B. die Art der zugrundeliegenden Programmzeile und keine Gewichtung vorhanden. Ein betrachtetes Programm besteht demnach in der Abstraktion nur noch aus einer ungeordneten Menge von Punkten. Die unterschiedlichen Arten der zeilenmäßigen Größenbestimmung liegen in der genauen Definition, was als Zeile aufgefaßt wird, d.h. welche Zeilen in den Container überführt und welche übergangen werden.
  - *Zyklomatische Zahl*: Hierbei wird jedes im Quelltext vorkommende Kommando, das Einfluß auf den Kontrollfluß hat (z.B. IF, SWITCH, DO, LOOP etc.), zu einem Punkt oder Knoten abstrahiert und in einen für jede Funktion existierenden Container hinzugefügt. Wie bereits bei LOC existieren zu einem Punkt keine weitergehenden Informationen. Ein betrachtetes Programm besteht demnach in der Abstraktion nur noch aus einer ungeordneten Menge von Punkten, die stellvertretend für die verschiedenen, den Kontrollfluß modifizierenden Anweisungen sind; die häufig für die zyklomatische Zahl vorgestellte Abstraktion der Kontrollflußgrapherstellung erlaubt zwar

die Betrachtung des Merkmals Kontrollfluß, ist aber für die zyklomatische Zahl zu detailreich. Aus dieser einfachen, expliziten Sicht wird deutlich, daß die Verwendung der zyklomatischen Zahl als Komplexitätsmaß ungeeignet ist, da bzgl. eines speziellen Merkmals sehr stark abstrahiert wird und viele ebenfalls auf die Eigenschaft Komplexität einwirkende Faktoren ignoriert werden.

  - *WMC*: Hierbei wird jede im Quelltext vorkommende Klasse zu einem Container abstrahiert, dessen Inhalt durch Größenabstraktionen der Methoden der Klasse gegeben ist. Die unterschiedlichen Größenabstraktionen der unterschiedlichen Methoden werden dabei derart gemischt, daß eine separate Betrachtung einzelner Größenabstraktionen der unterschiedlichen Methoden nicht mehr möglich ist.

- *Kopplungsabstraktionen*: Hierbei werden spezielle Systemteile extrahiert und spezielle Verbindungen zwischen ihnen modelliert, d.h. Kopplungsabstraktionen erstellen einen Graphen mit unterschiedlichen Knotenarten und Kantenarten. Dies bedeutet für einige typische, in Kapitel 3.2.2 aufgezeigten Maße folgende dahinter liegende Abstraktionsfunktionen (wie bereits bei den Größenabstraktionen wird hier die maximal abstrahierende Funktion beschrieben):
  - *Fan-in*: Hierbei werden alle Module und alle global definierten Variablen als Knoten extrahiert. Anschließend werden zwischen jeweils zwei Modulen die Kanten durch die gerichteten Benutztbeziehungen $\texttt{call}_i$ modelliert und zwischen Modulen und Variablen die gerichteten Beziehungen $\texttt{read}_i$ als Kanten eingetragen.
  - *CBO*: Hierbei werden alle Klassen als Knoten extrahiert und zwischen ihnen gerichtete Benutztbeziehungen als Kanten eingetragen. In der Abstraktion besitzen die Klassen keine Methoden, Attribute oder Vererbungen mehr, sondern lediglich die eingetragenen Benutzungsrelationen zu anderen Klassen.
  - *NOP*: Hierbei werden alle Klassen als Knoten extrahiert und zwischen ihnen als Kanten die gerichteten Vererbungsbeziehungen eingetragen. Wie bereits bei CBO besitzen die Klassen keinen Inhalt mehr, sondern lediglich die eingetragenen Vererbungsrelationen zu anderen Klassen.
- *Kohäsionsabstraktionen:* Die Kohäsionsabstraktionen sind denen der Kopplungsabstraktionen sehr ähnlich: Auch hier werden Graphen, bestehend aus verschiedenen Knotenarten und Kantenarten, erzeugt. Die Art der betrachteten Kanten, die Aufschluß über den Grad der Zusammengehörigkeit der extrahierten Systemteile geben sollen, können hier allerdings vielfältiger sein (vgl. auch Kapitel 7). Für einige typische, in Kapitel 3.2.3 aufgezeigten Maße resultieren daraus folgende dahinter liegende Abstraktionsfunktionen (erläutert werden wieder die maximal abstrahierenden Funktionen):
  - *SFC*: Hierbei werden alle Ausgabevariablen einer Funktion und alle vorkommenden Daten-Token als Knoten extrahiert. Anschließend werden zwischen Ausgabevariablen und Daten-Token die gerichteten Kanten „hat Einfluß auf“ hinzugefügt.
  - *LCOM*: Hierbei werden alle Methoden und Attribute einer Klasse als Knoten extrahiert und von Methoden zu Attributen gerichtete Benutztbeziehungen als Kanten eingetragen. Für das erweiterte LCOM-Maß werden zusätzliche Kanten durch gerichtete Benutztbeziehungen zwischen den Methoden modelliert (vgl. Kapitel 3.2.3).

Für ein gegebenes Produkt gibt es nicht die absolute, richtige Abstraktion, sondern lediglich eine (oder mehrere) für die Betrachtung eines Merkmals geeignete. So wie ein Produkt mehrere Merkmale besitzt, besitzt jedes Produkt mehrere Abstraktionen. Gleichzeitig müssen sich die konkreten Abstraktionen für unterschiedliche Produkte keineswegs immer unterscheiden, d.h. die Funktion *abs* muß keineswegs eineindeutig sein. So sind z.B. für ein gegebenes Produktmodell in Form eines Vererbungsbaums beliebig viele, unterschiedliche Programme denkbar, die bzgl. dieser Abstraktion identische Produktmodelle bilden.

Die Abstraktion ist ein absolut notwendiges Requisit der Vermessung, selbst wenn die Abstraktion nach genügend Erfahrung implizit wird. Neben der Erstellung dieser Abstraktionen ist die Findung von aussagekräftigen Relationen darauf die schwierigste Aufgabe, bevor sinnvoll vermessen werden kann, und wird im nächsten Abschnitt behandelt.

4.1.2 Relationen auf Produktmodellen

Ist das Produktmodell vollständig, eindeutig, konsistent und verständlich definiert, so müssen im nächsten Schritt die für die Analyse des betrachteten Merkmals relevanten Relationen identifiziert werden. Dieser Schritt ist mit äußerster Sorgfalt vorzunehmen, da die durch anschließendes Messen erhaltenen Ergebnisse lediglich auf den bzgl. des Produktmodells identifizierten Relationen beruhen dürfen[9]. Ein wichtiger Schritt bei der Identifikation von Relationen ist die genauere Untersuchung des Typus des betrachteten Merkmals, da dieses Indizien für die Art der zu extrahierenden Relationen gibt.
Merkmale können wie folgt klassifiziert werden ([FaHa84], S. 8ff und S. 376ff):

- *qualitative Merkmale*, die lediglich in einigen, eventuell geordneten, Kategorien vorliegen, und
- *quantitative Merkmale*, die in geordneten Intervallen und eventuell in Verhältnissen vorliegen.

Qualitative Merkmale können verfeinert werden in *nominalskalierte binäre*, *nominalskalierte mehrstufige* und *ordinalskalierte* Merkmale; quantitative Merkmale in *intervallskalierte* und *verhältnisskalierte* Merkmale. Die Klassifizierung eines zu betrachtenden Merkmals geschieht nicht zum Selbstzweck, sondern ermöglicht die gezielte Identifizierung von Relationen auf dem Produktmodell.

Relationen auf Produktmodellen mit qualitativen Merkmalen
Die einfachste Klasse von Merkmalen sind *nominalskalierte binäre Merkmale*, die genau zwei Ausprägungen haben: *Merkmal vorhanden* und *Merkmal nicht vorhanden*. Zwischen den beiden Ausprägungen existiert keinerlei Ordnung oder Wertung. Die einzige mögliche Relation, die für solche Merkmale sinnvoll ist, ist die Gleichheitsrelation $\doteq$, mittels derer bestimmt werden kann, ob zwei Produkte bzgl. des betrachteten Merkmals gleich sind, d.h. ob bei beiden das Merkmal vorhanden oder bei beiden das Merkmal nicht vorhanden ist.
Ein Beispiel eines nominalskalierten, binären Merkmals ist das Merkmal *abstrakt* einer Klasse: Eine Klasse kann entweder abstrakt oder nicht abstrakt sein, d.h. es ist binär; außerdem scheint keine Ordnung bzgl. der beiden Ausprägungen sinnvoll, d.h. es ist nominalskaliert.
Ebenfalls für die *nominalskalierten, mehrstufigen Merkmale* kann lediglich die Gleichheitsrelation $\doteq$ identifiziert werden, da zwar die Merkmale in mehreren Ausprägungen vorkommen, aber trotzdem keine sinnvolle Ordnung auf den verschiedenen, durch die Ausprägungen gegebenen Kategorien möglich ist.

[9] Dieser konservative Ansatz sagt aus, daß durch Messung konzeptionell keine neuen Ergebnisse hergeleitet werden können, die nicht im Vorfeld bereits innerhalb des Produktmodells gültig sind (vgl. *narrow view on measurement* in [SaEh92]). Die von Kriz und Zuse angeführte Möglichkeit, existierende „*Intelligence-Barriers*“, die im menschlichen Unvermögen begründet liegen, relevante Aussagen über den Problembereich vorzunehmen („*The human brain very often is not able to make relevant empirical statements*“, [Zuse98], S. 100), durch Messung zu überwinden, wird daher hier relativiert: Messung löst lediglich Problem-Barrieren, die nicht konzeptioneller, sondern technischer Natur sind. Diese eher als *Durchführbarkeits-Barrieren* beschreibbaren Probleme müssen für eine auf Messung basierende Lösung also konzeptionell bekannt und lösbar sein, können aber an praktischen Problemen scheitern (z.B. die Datenmengen sind zu groß, die zu vergleichenden Objekte sind nicht gleichzeitig betrachtbar, die manuelle Relationenüberprüfung ist zu aufwendig etc.).

Ein Beispiel eines nominalskalierten, mehrstufigen Merkmals ist das Merkmal *Autor* einer Datei: Jede Datei besitzt das Merkmal, wer die Datei erstellt hat. Die Anzahl unterschiedlicher Ausprägungen dieses Merkmals korrespondiert mit der Anzahl der am jeweiligen Projekt beteiligten Softwareentwickler. Auch hier sind nur Aussagen der Art „*Datei X und Datei Y sind von demselben Autor geschrieben worden*" oder „*Datei X und Datei Y sind von unterschiedlichen Autoren geschrieben worden*" sinnvoll.
Das Merkmal im letzten Beispiel kann, bei entsprechendem eine Ordnung ermöglichendem Wissen innerhalb der realen Welt aber auch als *ordinalskaliertes Merkmal* aufgefaßt werden: Die Mindestanforderung für diese Klasse von Merkmalen ist die Existenz einer *schwachen Ordnung* (engl.: *weak order*) über den Elementen des Produktmodells ([Kran et al.71], S. 14f; [FaHa84], S. 5ff); [Hend96], S. 70f): [10]

Eine binäre Relation $\Re$ auf den Elementen des Produktmodells $\mathcal{P}$ bildet eine *schwache Ordnung* auf $\mathcal{P}$, wenn mindestens gilt:

1) $\Re$ ist *konnex* (vollständig, total), d.h. $\forall a, b \in \mathcal{P}: (a \Re b) \vee (b \Re a)$ und

2) $\Re$ ist *transitiv*, d.h. $\forall a, b, c \in \mathcal{P}: (a \Re b) \wedge (b \Re c) \Rightarrow (a \Re c)$

Definition 17: Schwache Ordnung

Wenn z.B. für das Merkmal Autor eine schwache Ordnung durch die Beobachtung der relativen Einhaltung von Programmierrichtlinien eines jeden Autors existiert (z.B. „*Autor X wendet die Programmierrichtlinien konsequenter an als Autor Y*"), so kann – wenn diese Ordnung konnex und transitiv ist – dieses Merkmal als ordinalskaliert bezeichnet werden.

Ein Großteil der empirischen Forschungen ergibt zumindest eine schwache Ordnung für die betrachteten Entitäten ([Kran et al. 71], S. 14). Die übliche Darstellungsform solcher Ergebnisse sind sogenannte *Ranglisten* (engl.: *rankings)*, bei denen die Entitäten bzgl. der beobachteten Merkmale und deren Ordnung sortiert aufgelistet werden. Eine Beurteilung der Differenz zwischen zwei Plätzen innerhalb des Rankings ist nicht möglich.

Ein typisches Beispiel einer ordinalskalierten Eigenschaft innerhalb der Software-Entwicklung ist die Einordnung des zugrundeliegenden Prozesses innerhalb des CMM-Modells (vgl. Abbildung 14):
Es handelt sich hierbei um eine Eigenschaft und nicht um ein Merkmal, da Prozeßreife entsprechend des CMM-Modells einer von Menschen definierten Bedeutung bedarf. Erst durch feste Setzungen innerhalb des CMM-Modells besitzt diese Eigenschaft einen ordinalen Charakter und ist folglich meßbar. Die Eigenschaft Prozeßreife besitzt fünf Ausprägungen, die bzgl. der Reife wie in Abbildung 14 geordnet werden können (vgl. [Hump89]). Eine Aussage über die Größe der Intervalle zwischen den Reifegraden ist nicht möglich.

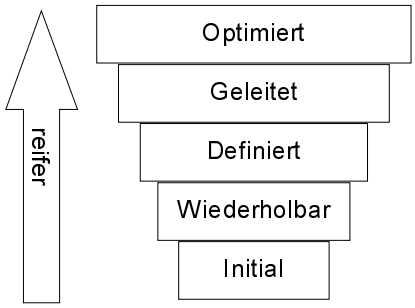


Abbildung 14: Geordnete Ausprägungen des Merkmals Prozeßreife entsprechend des CMM-Modells

[10] Die ebenfalls häufig aufgeführte Forderung für ein ordinalskaliertes Merkmal nach einer Äquivalenzrelation und einer einfachen Ordnung (z.B. [Pfan71], S. 29) kann aus einer schwachen Ordnung gewonnen werden ([Kran et al.71], S. 15f).

Relationen auf Produktmodellen mit quantitativen Merkmalen
So wie ordinalskalierte Merkmale mehr Informationen liefern als nominalskalierte, erlauben quantitative Merkmale eine genauere Analyse als qualitative Merkmale. Letztere unterscheiden sich in *intervallskalierte* und *verhältnisskalierte Merkmale*. Die größeren Analysemöglichkeiten quantitativer Merkmale erfordern ein sehr viel tiefergehendes Verständnis des Produktmodells (bzw. der ihnen zugrunde liegenden Entitäten), als dies für qualitative Merkmale der Fall ist. Dies äußert sich durch erweiterte Anforderungen an diese Klasse von Merkmalen: Zusätzlich zu einer binären Relation $\Re_1$, die den Anforderungen an eine schwache Ordnung auf den Elementen des Produktmodells genügt (vgl. ordinalskalierte Merkmale), muß mindestens eine quaternäre Relation $\Re_2$ existieren, die für Paare des Produktmodells eine schwache Ordnung definiert (vgl. [Zuse98], Kapitel 4.13.4), d.h.

1. $\forall a, b, c, d \in \mathcal{P}: (a \times b) \Re_2 (c \times d) \vee (c \times d) \Re_2 (a \times b) \wedge$
2. $\forall a, b, c, d, e, f \in \mathcal{P}: (a \times b) \Re_2 (c \times d) \wedge (c \times d) \Re_2 (e \times f) \Rightarrow (a \times b) \Re_2 (e \times f)$

Innerhalb des Produktmodells muß folglich ein Verständnis über Differenzen und deren Ordnung existieren. Ein berühmtes Beispiel für ein intervallskaliertes Merkmal ist die Temperatur (vgl. [FePf96], Kapitel 2.3.3): Temperaturen können sowohl einzeln geordnet werden („*heute ist es wärmer als gestern*“) als auch in Paaren („*von vorvorgestern zu vorgestern hat es sich weniger aufgewärmt als von gestern zu heute*“). Ein anderes Beispiel für ein intervallskaliertes Merkmal ist die Zeit: Auch hier ist eine schwache Ordnung definiert (z.B. „*später als*“) und es ist ebenfalls möglich, Zeitintervalle, die als *Dauer* bezeichnet und durch die Differenz von Start- und Endzeit gegeben sind, zu ordnen (z.B. „*hat mindestens so lange gedauert wie*“). Typisch für intervallskalierte Merkmale ist, daß Aussagen über Verhältnisse wie z.B. zwischen Temperaturen „*heute ist es doppelt so warm wie gestern*“ nicht möglich sind.
Für den Bereich der Softwaremaße wird allgemein angenommen, daß intervallskalierte Merkmale wenig intuitiv sind ([Zuse98], S. 144) und daher in diesem Bereich deutlich seltener vorkommen als nominal- und ordinalskalierte Merkmale ([FePf96], S. 50). Im weiteren Verlauf dieser Arbeit wird aber gezeigt werden, daß mittels einer generischen Merkmalsidentifikation gerade intervallskalierte Merkmale erzeugt werden, die ein sehr aussagekräftiges Vermessen von Software ermöglichen und für alle in Kapitel 3.3 vorgestellten Prozesse effektiv und effizient verwendet werden können.
Die Klasse der *verhältnisskalierten Merkmale* erfordert für das Produktmodell, daß Verhältnisse zwischen den Merkmalen interpretierbar sind. Erst wenn dies der Fall ist, sind Aussagen der Art „*a ist n-mal größer als b* “ möglich. Im Gegensatz zu den oben aufgeführten Merkmalsklassen benötigen verhältnisskalierte Merkmale folglich zusätzlich zu den Relationen noch eine binäre Operation $\circ$, die jedem Paar a, b von Elementen eines Produktmodells $\mathcal{P}$ ein $a \circ b \in \mathcal{P}$ zuweist (*Verkettungsoperation*, [FaHa84], S. 8). Merkmale, für die eine solche Operation innerhalb des Produktmodells sinnvoll hergeleitet werden können, werden als *extensive Merkmale* bezeichnet. Der Grundgedanke von extensiven Merkmalen ist die Übertragung der archimedischen Eigenschaft der reellen Zahlen auf das Produktmodell ([FaHa84], S.8f). Innerhalb $\mathbb{R}$ ist diese Eigenschaft wie folgt definiert:

Archimedische Eigenschaft für reelle Zahlen: Sind $x, y \in \mathbb{R}$ und ist $x>0$, so gibt es ein $n \in \mathbb{N}$ mit $n*x>y$

Definition 18: Archimedische Eigenschaft für reelle Zahlen entsp. ([FaHa84], S. 9)

Die Notwendigkeit einer binären Operation für verhältnisskalierte Merkmale ergibt sich aus der Forderung, Aussagen über ein Vielfaches einer Entität („$n*x$“) machen zu können. Der Aussage „*a ist dreimal so lang wie b*“ liegt eine implizite binäre Operation zugrunde, die durch zweifache

Anwendung auf *a*, d.h. (*a* o (*a* o *a*)), ein Ergebnis ergibt, das wiederum mit einer anderen Entität bzgl. des betrachteten Merkmals ordnungsrelativ verglichen werden können muß („>*y*“).
Eine schwierige Frage ist häufig das Verständnis der durch die Operation neu geschaffenen Entität: Muß (*a* o *a*) als Entität in der betrachteten Welt vorkommen (oder genauer: muß es in der betrachteten Welt eine Entität geben, deren Produktmodell durch (*a* o *a*) gegeben ist), hypothetisch vorstellbar sein oder genügt die bloße Definition der binären Operation ohne direktes Pendant in der Realität? Diese Frage ist auch aus meßtheoretischer Sicht sehr interessant (vgl. [SaEh92]) und bedeutet für die Vermessung von Software, ob eine Operation auf zwei Softwaremodulen wieder ein Softwaremodul ergeben muß (muß z.B. die Verknüpfung zweier Klassen eines objektorientierten Systems wieder eine Klasse ergeben? Wie sieht diese Operation dann aus?) So werden z.B. in [Zuse98] eine Vielzahl von Operationen vorgestellt, um die Verhältnisskalierung der betrachteten Merkmale zu belegen. Andere Autoren bezweifeln den Sinn solcher Operationen, da sie nur selten eine Entsprechung in der Realität besitzen: „[...] *we rarely concatenate programs*.“ ([GuTaWe93], S. 164).
Da sich allerdings die Notwendigkeit von Operationen bereits aus der Formulierung der Relationen ergibt, genügt deren genaue Betrachtung. Die Frage nach Operationen ist daher zweitrangig und kann, falls dies die Relationen verlangen, als Gedankenexperiment formuliert werden, um ein vertieftes Verständnis von den betrachteten Merkmalen zu erhalten und möglicherweise Relationen darauf zu falsifizieren oder zu bestätigen. Geschieht diese Betrachtung nicht explizit, so sind bei gültigen Relationen, die das Merkmal als verhältnisskaliert charakterisieren, die notwendigen Operationen implizit mit vorausgesetzt: „*Concatenation operations are not artificial, they are simply there* “ ([Zuse98], S. 298). Für die weitere Arbeit werden sie daher nicht explizit angesprochen. Darüber hinaus wird diese Arbeit zeigen, daß bereits ordinal- und intervallskalierte Merkmale effektiv für den Bereich der Softwareproduktvermessung eingesetzt werden können. Diese Vereinfachung trägt ebenfalls wesentlich zur Lösung des Wert-Interpretations-Problems und des Meßtheorie-Anwendungs-Problems bei (vgl. Kapitel 3.6.2).

4.1.3 Relationensysteme

Sowohl die Untersuchung der betrachteten Merkmale und die dadurch bestimmte Erstellung von Produktmodellen als auch die Identifikation von Relationen auf den Merkmalen sind unabdingbare Voraussetzungen jeden Messens; speziell die sorgfältige Identifikation von Relationen kann sehr aufwendig sein: „*This is probably the most critical step* [...]“ ([Baker et al.90b], S. 253). Je aussagekräftiger allerdings die Merkmale und die Relationen gewählt sind, desto aussagekräftigere Ergebnisse können durch das Messen erzielt werden (vgl. Abschnitt 4.1.2). Das Messen stützt sich immer auf Aussagen, die prinzipiell in der realen Welt auch ohne Messungen vorgenommen werden können: „*Measurement is ontologically committed (i.e., rooted in and, hence, grounded by objective reality)* “ ([Berk92], S. 182).
Da die gefundenen Relationen und die dazugehörigen Produktmodelle kausal zusammengehören, werden beide Mengen häufig durch folgende Definition zusammengehalten:

> Sei $\mathcal{A}$ eine Menge von Objekten und $\mathcal{R}_1$, ..., $\mathcal{R}_n$ seien auf $\mathcal{A}$ definierte Relationen. Dann heißt das System $<\mathcal{A}; \mathcal{R}_1, ..., \mathcal{R}_n>$ *relationales System* oder auch *Relationensystem*.

Definition 19: Relationales System, nach [Pfan71], S. 18

Für eine Klassifizierung der Relationensysteme kann die Art der betrachteten Objekte innerhalb der Menge $\mathcal{A}$ wie folgt unterteilt werden:

- *Empirische Objekte*: Früher wurden empirische Objekte häufig als beobachtbare Objekte bezeichnet. Diese Bezeichnung greift heute allerdings zu kurz, da viele Objekte wie z.B. Zugriffsrechte oder Log-Files lediglich auf Rechnern abgelegt und nicht direkt beobachtbar sind (vgl. [Zuse98], S. 92). Darüber hinaus wird fälschlicherweise angenommen, daß die Objekte aus $\mathcal{A}$ in der Realität vorkommen. Es wurde allerdings gezeigt, daß Relationen nur

auf Modellen der Objekte der Realität definiert werden können (vgl. Abschnitt 4.1.1). Unter einem empirischen Objekt wird daher im folgenden ein innerhalb eines Produktmodells gegebenes Element verstanden, das die Abstraktion eines in der realen Welt existierenden und – direkt oder indirekt – erfahrbaren Objekts ist.

- *Formale Objekte*: Entsprechend der Definition eines relationalen Systems können die Elemente aus $\mathcal{A}$ ebenfalls Zahlen oder Vektoren sein. Die angegebenen Relationen können dann z.B. durch zweistellige Relationen wie „>" oder „<" gegeben sein. Typisch für diese formalen Objekte ist, daß sie keiner Modellierung benötigen, um auf ihnen Relationen definieren zu können.

Entsprechend dieser Klassifizierung können Relationensysteme klassifiziert werden in

- *empirische Relationensysteme*, bei denen $\mathcal{A}$ aus empirischen Objekten besteht, und
- *formale Relationensysteme*, bei denen $\mathcal{A}$ aus formalen Objekten besteht.

Im folgenden werden mittels spezieller Funktionen empirische Relationensysteme, deren empirische Objekte für den Bereich der Software-Vermessung durch die Elemente der Produktmodelle gegeben sind (vgl. Kapitel 3.2), auf formale Relationensysteme abgebildet. Die darauf aufbauende Definition von Skalen stellt die meßtheoretische Formulierung jeden Meßvorgangs dar.

4.2 Skalen

Nachdem ein empirisches Relationensystem (ERS) mit aussagekräftigen Relationen gefunden wurde, ist eine weitere Voraussetzung eines Meßvorgangs das Identifizieren eines entsprechenden formalen Relationensystems (FRS): In ihm soll sowohl jede im ERS vorkommende Entität eine Entsprechung besitzen als auch jede im ERS identifizierte Relation ein Pendant haben.
Die Grundidee des Messens besteht darin, Eigenschaften von Merkmalsausprägungen aufgrund einer Analyse im FRS bewerten zu können und diese Ergebnisse auf das ERS zu übertragen. Damit dies möglich ist, muß die Funktion, die als Definitionsbereich das ERS und als Bildbereich das FRS besitzt, die sogenannte *Repräsentationsbedingung* (vgl. [FePf96], S. 31f) erfüllen:

> Eine Funktion $\mu: \mathcal{A} \rightarrow \mathcal{B}$ von einem ERS $<\mathcal{A}; \mathcal{R}_{\mathcal{A}_1}, ..., \mathcal{R}_{\mathcal{A}_n}>$ in ein FRS $<\mathcal{B}; \mathcal{R}_{\mathcal{B}_1}, ..., \mathcal{R}_{\mathcal{B}_n}>$, für die gilt:
> - $\forall a \in \mathcal{A} \exists\ b \in \mathcal{B} : \mu(a)=b$ und
> - $\forall a_1, ..., a_{r_i} \in \mathcal{A}$ gilt: $(a_1, ..., a_{r_i}) \in \mathcal{R}_{\mathcal{A}_i} \Leftrightarrow (\mu(a_1), ..., \mu(a_{r_i})) \in \mathcal{R}_{\mathcal{B}_i}$ wobei $\mathcal{R}_{\mathcal{A}_i}$ und $\mathcal{R}_{\mathcal{B}_i}$ r_i-stellige Relationen sind,
>
> ist ein *Maß*.

Definition 20: Maß (entspr. [FaHa84], S. 6f; [Pfan71], S. 22f)

Neben der bloßen Abbildung der Elemente aus $\mathcal{A}$ nach $\mathcal{B}$ ist entsprechend Definition 20 also zusätzlich die *Relationserhaltung* gefordert. Für eine zweistellige Relation $\mathcal{R}_{\mathcal{A}_i}$ bedeutet dies, daß zwei Elemente a_1 und a_2 genau dann Element der Relation $\mathcal{R}_{\mathcal{A}_i}$ sind, wenn die Abbilder $b_1 = \mu(a_1)$ und $b_2 = \mu(a_2)$ Element der Relation $\mathcal{R}_{\mathcal{B}_i}$ sind. Besonders wichtig ist die Bedingung „*genau dann wenn*" bzw. $\Leftrightarrow$ in Definition 20, da μ anderenfalls nicht als Maß verwendet werden darf (vgl. [Pfan71], S. 26f).
Die Aufgabe eines Maßes ist folglich, ohne Einschränkung der Aussagekraft innerhalb des ERS die empirischen Objekte aus $\mathcal{A}$, die Elemente beliebig komplexer Produktmodelle sein können, auf einfacher verarbeitbare, formale Objekte aus $\mathcal{B}$ abzubilden und gleichzeitig die i.d.R. nur schwierig durchzuführende Relationsbestimmung (vgl. Abschnitt 4.1.2) mittels einfacher zu bestimmenden Relationen auf formalen Objekten nachzubilden.

Da jedem sinnvollen Meßvorgang ein ERS, ein FRS und ein Maß zwischen den beiden zugrunde liegt, also jedes Messen durch das Tripel (ERS, FRS, μ) spezifiziert werden kann, werden die drei Elemente wie folgt zusammengefaßt:

> Existiert zu einem gegebenen ERS $<\mathcal{A}; \mathcal{R}_{\mathcal{A}_1}, ..., \mathcal{R}_{\mathcal{A}_n}>$ ein FRS $<\mathcal{B}; \mathcal{R}_{\mathcal{B}_1}, ..., \mathcal{R}_{\mathcal{B}_n}>$ und ein Maß μ von $\mathcal{A} \rightarrow \mathcal{B}$, so wird das Tripel $(<\mathcal{A}; \mathcal{R}_{\mathcal{A}_1}, ..., \mathcal{R}_{\mathcal{A}_n}>, <\mathcal{B}; \mathcal{R}_{\mathcal{B}_1}, ..., \mathcal{R}_{\mathcal{B}_n}>, \mu)$ als *Skala* bezeichnet.

Definition 21: Skala (entspr. [Hend96], S. 68f)

Aus der Sicht des Anwenders, der nach vollständiger Erstellung eines ERS dessen Elemente vermessen will, d.h. als notwendige Voraussetzung dafür eine Skala ermitteln will, stellen sich diesbezüglich folgende Fragen:

1. Gibt es für jedes ERS ein FRS, das die Repräsentationsbedingung vollständig, d.h. für alle Objekte und Relationen im ERS erfüllt?
2. Wenn für ein ERS mehrere FRS existieren, die die Repräsentationsbedingung erfüllen, gibt es dann Kriterien, nach denen ein FRS favorisiert werden sollte?

Beide Fragen stellen das zentrale Aufgabenfeld der formalen Meßtheorie dar ([SaEh92]):

1. Die Beschäftigung mit Bedingungen, die innerhalb des ERS erfüllt sein müssen, damit die Existenz von Skalen mit bestimmten Objekten innerhalb des FRS (z.B. mit Elementen aus $\mathbb{R}$) garantiert werden kann, mündet in sogenannte *Repräsentationstheoreme* ([SaEh92], S. 5f); die konkrete Problematik wird als *Repräsentationsproblem* bezeichnet ([FePf96], S. 46).
2. Die Beschäftigung mit unterschiedlichen Skalen für ein gegebenes ERS, die jeweils die Repräsentationsbedingung erfüllen, und die Möglichkeit von Transformationen zur Überführung eines FRS in ein anderes, münden in sogenannte *Eindeutigkeitstheoreme* ([SaEh92], S. 6f); die konkrete Problematik wird als *Eindeutigkeitsproblem* bezeichnet ([FePf96], S. 46).

Beide Probleme sind nicht auf den Bereich der Softwarevermessung beschränkt, sondern treffen alle Wissenschaftsbereiche, die mit Messungen konfrontiert sind. Trotzdem sollen an dieser Stelle nur die für den Bereich der Softwarevermessung relevanten Problemaspekte erläutert werden:
Aufgrund des noch geringen Alters des Wissenschaftsbereichs Software-Engineering sind die ERS in der Regel sehr einfach gehalten. In vielen Fällen bereitet bereits die Identifikation von ordinalskalierten Merkmalen große Probleme. Für die wenigen Relationen, die im FRS erhalten werden müssen, existieren in der Regel bereits Repräsentationstheoreme, die die Existenz einer geeigneten Skala sicherstellen (vgl. z.B. *Cantor-Theorem* in [Kran et al.71], S. 39f). Das Repräsentationsproblem stellt sich daher innerhalb der Softwarevermessung als überwindbar heraus: So sind z.B. für alle Messungen innerhalb des Meßkatalogs in [LoKi94] formale Relationensysteme mit Objekten aus $\mathbb{R}$ (oder Teilmengen davon) und einfachen Relationen wie „>“ oder „<“ verwendet.

Das Eindeutigkeitsproblem läßt sich anhand der Klassifizierung von Merkmalen (vgl. Abschnitt 4.1.2) ebenfalls lösen, soll an dieser Stelle aber, da die Konsequenzen häufig nicht entsprechend berücksichtigt werden und dadurch die Meßergebnisse fehlerhaft werden können, detaillierter erläutert werden: Entsprechend der Merkmalsklassifizierung in nominalskalierte, ordinalskalierte, intervallskalierte und verhältnisskalierte Merkmale können entsprechende *Skalentypen* definiert werden. Diese Skalentypen wurden erstmals von Stevens eingeführt [Stev46]. Für jeden Skalentyp sind dabei Klassen von *zulässigen Transformationen* gegeben: Diese überführen ein Maß μ_{alt} derart in ein neues Maß μ_{neu}, so daß μ_{neu} dieselben Relationen im ERS auf dieselben Relationen im FRS abbildet wie μ_{alt}, d.h. die Repräsentationsbedingung bleibt erfüllt. Die Klasse der zulässigen Transformationen definiert also den Skalentyp ([Zuse98], S. 132). Aufgrund dieses

Skalentyps läßt sich sehr anschaulich sowohl die Menge von statistischen Techniken, die für eine Skalenklasse sinnvoll angewendet werden können, als auch diejenige Menge von statistischen Techniken, die zu sinnlosen, d.h. nicht auf das ERS zurück übertragbaren Ergebnissen führen, ableiten. Damit erlaubt die Beschäftigung mit Skalentypen eine Definition des Terminus „*sinnvolle Aussagen im ERS*“ und behandelt damit das dritte große Aufgabenfeld der formalen Meßtheorie, das *Aussagekraftsproblem* (gelegentlich auch bezeichnet als *Bedeutsamkeitsproblem*, z.B. [BoDö95]):

> Eine auf Meßwerten basierende Aussage innerhalb des ERS ist *sinnvoll*, wenn ihre Herleitung invariant gegenüber zulässigen Transformationen des jeweiligen Skalentyps auf dem zugrundeliegenden FRS ist.

Definition 22: Sinnvolle Aussagen im ERS (nach [Robe79], S. 58)

Damit erschließt diese Definition zwar keine neuen Erkenntnisse über mögliche, sinnvolle Aussagen (da der aus dem FRS extrahierbare Sinngehalt nach wie vor nicht die Menge der im ERS definierten, sinnvollen Relationen übersteigen darf: „*But, be careful with these scale types, look for the empirical conditions assumed by the measure.*“, [Zuse98], S. 658), erlaubt aber eine zusätzliche Überprüfungsmöglichkeit: Eine verwendete statistische Technik kann durch eine auf dem FRS basierende Widerlegung (oder Bekräftigung) häufig mit weniger Aufwand bzgl. ihres Sinngehalts auf dem jeweiligen Skalenniveau untersucht werden.
Im folgenden soll jede von Stevens eingeführte Skalenklasse [Stev46] und die darin definierte Menge von zulässigen Transformationen kurz erläutert und bzgl. der darin möglichen statistischen Techniken vorgestellt werden. Darüber hinaus wird im Abschnitt 4.2.5 die Absolutskala beschrieben[11]. Die Beschäftigung mit Skalen hilft wesentlich, daß Maß-Definitionsproblem (vgl. Kapitel 3.6.2) zu lösen, da ein Großteil dieses Problems im Fehlen eben einer Skala und der fehlenden Einordnung innerhalb der folgenden Skalenklassifizierung liegt.

4.2.1 Nominalskala

Eine Nominalskala repräsentiert lediglich ungewertete Unterschiede zwischen den vermessenen Objekten ([SaEh92], S. 3). Die betrachteten Merkmale innerhalb dieses Skalentyps sind lediglich nominalskaliert (binär oder mehrstufig, vgl. Abschnitt 4.1.2). Das Maß, das einer Entität $a \in$ ERS ein Element $b \in$ FRS zuweist, muß also lediglich die Unterscheidbarkeit von bzgl. des betrachteten Merkmals unterschiedlichen Entitäten sicherstellen. Die Menge der zulässigen Transformationen innerhalb des FRS umfaßt daher jede eineindeutige Funktion.
Aufgrund der Definition 22 können folglich keinerlei sinnvolle Aussagen auf einem ERS einer Nominalskala gemacht werden, die ausschließlich auf der Größe der Elemente des FRS basieren, da diese mittels einer beliebigen eineindeutigen Funktion verändert werden können. Einzig allein die Ergebnisse von Häufigkeitsanalysen als statistische Verfahren können durchgeführt und sinnvoll auf das ERS zurück übertragen werden (z.B. „wieviel Klassen sind abstrakt“) ([FePf96], S. 59f).

4.2.2 Ordinalskala

Eine Ordinalskala repräsentiert eine Ordnung auf den vermessenen Objekten ([SaEh92], S.3). Die betrachteten Merkmale innerhalb dieses Skalentyps müssen als ordnungsprägende Elemente folglich mindestens ordinalskaliert sein (vgl. Abschnitt 4.1.2). Das Maß, das den Elementen a_1, $a_2 \in$ ERS die Elemente b_1, $b_2 \in$ FRS zuweist, muß also lediglich die bzgl. des betrachteten Merkmals gegebene Ordnung zwischen den Elementen a_1, a_2 aus dem ERS im FRS beibehalten.

[11] Die Absolutskala ist entgegen [Zuse98] (vgl. S. 43 und S. 660) nicht Teil der von Stevens eingeführten Skalentypen (vgl. [Steve.46], Tabelle 1, S. 678). Aufgrund ihrer sehr restriktiven Klasse der zulässigen Transformationen, die das Messen als einfaches Zählen erscheinen läßt, wird sie auch nur bedingt in der Meßtheorie behandelt (vgl. z.B. [SaEh92]).

Die Menge der zulässigen Transformationen umfaßt daher jede streng monoton steigende Funktion[12] f, da

$(a_1 \mathfrak{R}_{ERS} a_2)$
$\Leftrightarrow (\mu(a_1) \mathfrak{R}_{FRS} \mu(a_2))$
$\Leftrightarrow (f(\mu(a_1)) \mathfrak{R}_{FRS} f(\mu(a_2)))$
(letztere Äquivalenz ist wahr, da μ auf formale Objekte abbildet (z.B. $\mathbb{R}$) mit bekannten Relationen (z.B. „≤“) für die gilt: $(x \le y) \Leftrightarrow (f(x) \le f(y))$, falls f streng monoton steigend ist.)

Aufgrund der Definition 22 können folglich nur sinnvolle Aussagen auf einem ERS einer Ordinalskala gemacht werden, die auf der relativen Ordnung der Elemente des FRS basieren. Statistische Techniken für diesen Skalentyp sind daher – zusätzlich zu den für die Nominalskala gültigen – die Berechnung des Medians, verschiedene Perzentile (z.B. Quartile) und nicht-parametrische Korrelationskoeffizienten wie z.B. die von Kappa, Spearman oder Kendall Tau (vgl. z.B. [Zuse98]).

4.2.3 Intervallskala

Eine Intervallskala repräsentiert Intervalle zwischen den vermessenen Objekten ([SaEh92], S. 3). Um dies zu ermöglichen, müssen die innerhalb dieses Skalentyps betrachteten Merkmale mindestens intervallskaliert sein (damit überhaupt ein Verständnis von Intervallen zwischen Merkmalen möglich ist, vgl. Abschnitt 4.1.2). Das Maß, das den Entitäten $a_1, a_2, a_3, a_4 \in$ ERS die Elemente $b_1, b_2, b_3, b_4 \in$ FRS zuweist, muß also die Ordnung zwischen den Differenzen invariant halten. Daraus folgt nicht nur die Forderung nach strenger Monotonie, sondern auch die nach Linearität. Die Menge der zulässigen Transformationen innerhalb des formalen Relationensystems ist also durch die positiv linearen Funktionen $f(x)=ax+b$ $(a>0)$ gegeben, da

$(a_1 \times a_2) \mathfrak{R}_{2ERS} (a_3 \times a_4)$
$\Leftrightarrow \mu(a_1 \times a_2) \mathfrak{R}_{2FRS} \mu(a_3 \times a_4)$
$\Leftrightarrow (a(\mu(a_1 \times a_2)+b) \mathfrak{R}_{2FRS} (a(\mu(a_3 \times a_4)+b)$
(letztere Äquivalenz ist wahr, da μ auf formale Objekte abbildet (z.B. $\mathbb{R}$) mit bekannten Relationen (z.B. „≤“) für die gilt: $(x \le y) \Leftrightarrow (ax+b) \le (ay+b)$, falls $a > 0$).

Aufgrund der Definition 22 können folglich nur sinnvolle Aussagen auf einem ERS einer Intervallskala gemacht werden, die auf der relativen Ordnung der Elemente des FRS oder auf deren Intervallen basieren. Statistische Techniken für diesen Skalentyp sind daher – zusätzlich zu den für die Ordinalskala gültigen – die Berechnung des arithmetischen Mittels, der Standardabweichung und einiger weiterer Korrelationskoeffizienten wie z.B. das von Pearson.

4.2.4 Rationalskala

Eine Rationalskala, häufig auch als Ratio- oder Verhältnisskala bezeichnet (vgl. z.B. [BoDö95]), repräsentiert Verhältnisse zwischen den vermessenen Entitäten ([SaEh92], S. 3). Um dies zu ermöglichen, müssen die innerhalb dieses Skalentyps betrachteten Merkmale verhältnisskaliert sein (damit überhaupt ein Verständnis von Verhältnissen zwischen Merkmalen möglich ist, vgl. Abschnitt 4.1.2). Das Maß, das den Entitäten $a_1, a_2 \in$ ERS die Elemente $b_1, b_2 \in$ FRS zuweist, muß also das Verhältnis zwischen a_1 und a_2 invariant halten. Daraus folgt nicht nur die Forderung nach strenger Monotonie und Linearität, sondern darüber hinaus die Forderung nach Einschluß des Nullpunktes (f(0)=0). Die Menge der zulässigen Transformationen innerhalb des formalen Relationensystems ist also durch die Ähnlichkeitsfunktionen des Typs

[12] Die in der Literatur (z.B. [FePf96], S. 53; [Zuse98], S. 134, S. 348) häufig anzutreffende, weniger restriktive Forderung einer monotonen Funktion genügt hier nicht. So ist zwar z.B. eine konstante Funktion monoton, zerstört aber die im ERS identifizierte Ordnung, d.h. sie ist keine zulässige Transformation auf dem FRS.

$f(x)=ax$ ($a>0$) gegeben, da für eine Operation O_{ERS} , die eine Entsprechung im FRS durch O_{FRS} besitzt, gilt:

$(a_1\ O_{ERS}\ a_2) = \mu(a_1)\ O_{FRS}\ \mu(a_2)$

$\Leftrightarrow (\ (a\ (\mu(a_1))\ O_{FRS}\ (a\ (\mu(a_2)))\ )$

(letztere Äquivalenz ist wahr, da μ auf formale Objekte abbildet (z.B. $\mathbb{R}$) mit bekannten Operationen (z.B. „/“) für die gilt: $(x / y) = (ax / ay)$).

Aufgrund der Definition 22 können folglich nur sinnvolle Aussagen auf einem ERS einer Rationalskala gemacht werden, die auf der relativen Ordnung der Elemente des FRS, auf deren Intervallen oder auf deren Verhältnisse basieren. Die statistischen Techniken für diesen Skalentyp umfassen daher – zusätzlich zu den für die Intervallskala gültigen – die Berechnung des geometrischen Mittels, des harmonischen Mittels und von prozentualen Verhältnissen.

4.2.5 Absolutskala

Eine Absolutskala repräsentiert zusätzlich zu allen Informationen einer Rationalskala absolute, interpretierbare Werte. Diese Skala wurde von Stevens nicht erwähnt [Stev46], läßt sich aber bzgl. der Klassifizierung entsprechend der zulässigen Transformationen herleiten: Da der Wert selbst interpretierbar ist, darf die zulässige Transformation ihn nicht verändern, d.h. es ist lediglich die identische Abbildung $f(x)=x$ erlaubt. Die Absolutskala umfaßt das alltägliche Phänomen des Zählens („*Counting is an example of an absolute scale*“, [Robe79], S. 64). Alle möglichen statistischen Techniken können auf der Absolutskala angewendet werden.

4.3 Validierung

Ein wesentlicher Vorteil der expliziten Beschäftigung mit der Meßtheorie, und dort besonders mit den erarbeiteten Zwischenergebnissen in Form der Skalen, ist die dadurch gegebene Möglichkeit der Maßvalidierung:

> Die *Maßvalidierung* ist der Prozeß der Sicherstellung, daß das Maß eine gültige, numerische Charakterisierung des betrachteten Merkmals darstellt.

Definition 23: Maßvalidierung (nach [Baker et al.90a], Def. 2.1)

Die Meßtheorie hilft darüber hinaus nicht nur bei der Gültigkeitsüberprüfung, sondern unterstützt ebenfalls das Erkennen der jeweiligen Grenzen einer Skala (vgl. [Zuse98], S. 264). Die auf der Meßtheorie basierende Maßvalidierung ist damit fundamentaler Bestandteil jedes erfolgreichen Einsatzes von Softwaremaßen.

Die Gültigkeit der numerischen Charakterisierung kann auf unterschiedlichen Ebenen stattfinden. Dies soll anhand der folgenden Abbildung 15 verdeutlicht werden, in der der Meßprozeß ausgehend von der Fragestellung, die das anschließende Vermessen motiviert, bis hin zur Rückübertragung der Meßergebnisse auf die Fragestellung vollständig dargestellt ist:

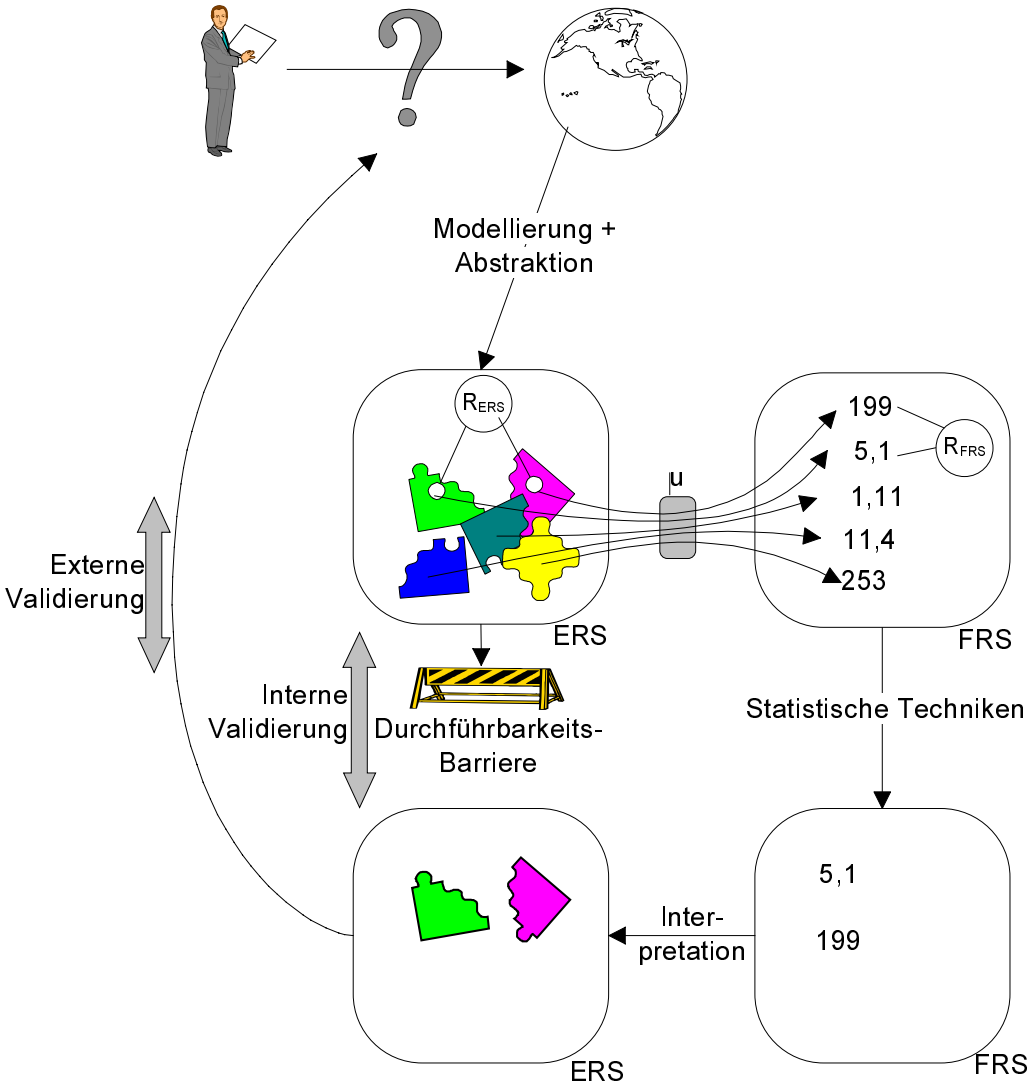


Abbildung 15: Modell vom Meßprozeß

Grundmotivation jedes Meßvorgangs sind Fragen über Entitäten der realen Welt. Um diese Fragen mittels Maßen beantworten zu können, müssen die Entitäten entsprechend vorbereitet werden. Dazu gehört im Falle der Softwareproduktvermessung die möglichst feingranulare Präzisierung mittels Merkmalen (vgl. Qualitätsmodelle in Kapitel 2) und die Erstellung eines relevanten Produktmodells zur Konzentration auf das betrachtete Merkmal zur Charakterisierung des Qualitätsmerkmals. Aufbauend auf diesem Produktmodell wird anschließend das empirische Relationensystem erstellt, dessen Elemente mittels eines Maßes μ auf ein formales Relationensystem relationenerhaltend abgebildet werden. Innerhalb dieses einfacheren Relationensystems können statistische Operationen durchgeführt werden, die jedoch innerhalb der Skalenklasse erlaubt sein müssen (vgl. Abschnitt 4.2). Das Ergebnis dieser statistischen Operationen wird anschließend durch Interpretation auf das empirische Relationensystem zurück übertragen. Diese Ergebnisse innerhalb des empirischen Relationensystems werden anschließend auf das Problem in der realen Welt angewendet, um die Ausgangsfrage bearbeiten zu können.

Anhand dieses Meßmodells können zwei verschiedene Arten der Gültigkeitsüberprüfung von Maßen vorgenommen werden (vgl. [KiPfFe95], [Baker et al.90a], [Zuse98]):

> Die *interne Validierung* von Softwaremaßen umfaßt die Sicherstellung, daß die Skala, die statistischen Operationen innerhalb des formalen Relationensystems und die dort stattfindende Interpretation konform zur Meßtheorie stattfinden.

Definition 24: Interne Validierung von Softwaremaßen

Anhand der Abbildung 15 bedeutet dies die Gültigkeit zwischen dem empirischen Relationensystem und den durch Messung erhaltenen Ergebnissen innerhalb des empirischen Relationensystems. Aufgrund ihrer Wurzeln in der Meßtheorie wird die interne Validierung häufig auch als *theoretische Validierung* bezeichnet ([KiPfFe95], [HaCoNi98]).
Eine weitere Gültigkeitsüberprüfung schließt die Betrachtung der realen Welt mit ein (vgl. [Zuse98], [KiPfFe95], [HaCoNi98]):

> Die *externe Validierung* von Softwaremaßen umfaßt die Überprüfung, inwieweit die Skala, die statistischen Operationen innerhalb des formalen Relationensystems und die dort stattfindenden Interpretationen die dem Messen zugrundeliegende Fragestellung beantworten helfen.

Definition 25: Externe Validierung von Softwaremaßen

Anhand der Abbildung 15 bedeutet dies die Relevanz der durch Interpretation des FRS erhaltenen Aussagen innerhalb des ERS für die Fragestellung in der realen Welt. Aufgrund der Vielzahl von Fragestellungen und Kontexten, in denen Maße verwendet werden, kann die externe Validierung nur statistisch für einige Fragestellungen innerhalb konkreter Kontexte vorgenommen werden. Diese Form der Validierung wird daher häufig auch als empirische Validierung bezeichnet ([KiPfFe95], [HaCoNi98]).
Im folgenden sollen beide Validierungsarten detailliert beschrieben werden, da ihre konsequente Anwendung hilft, das Wert-Interpretations-Problem (vgl. Kapitel 3.6.2) zu reduzieren.

4.3.1 Interne Validierung

Die interne Validierung von Softwaremaßen umfaßt die explizite Beschäftigung mit den in Abschnitt 4.1 und 4.2 gemachten Anforderungen an Maße. Ihre Relevanz ist daher besonders für zwei Anwendungsfälle bedeutsam:

- *Summative Validierung*: Ein bestehendes Maß soll angewendet werden, d.h. ausgehend von einer in den meisten Fällen unscharfen Definition des Maßes werden das zugrundeliegende Produktmodell, die darauf aufbauende Skala und die Menge der zulässigen statistischen Techniken rekonstruiert. Die Aufgabe der summativen Validierung besteht nicht nur aus der Nachreichung fehlender formaler Definitionen, sondern sie liegt ebenfalls in der Überprüfung, ob eine Skala in einem neuen Umfeld ohne Anpassung anwendbar ist: Da das ERS eventuell nur in einem bestimmten Kontext Gültigkeit besitzt (weil z.B. entsprechende Vorgaben eine Ordnung auf den Elementen definieren), muß dessen Gültigkeit innerhalb eines neuen Kontextes regelmäßig überprüft werden (da im Zielkontext eventuell diese ordnungsgebende Definition nicht etabliert ist). Damit trägt die interne Validierung der Aussage Rechnung, daß „*a measure only can be validated related to a certain environment. If a measure is valid in one environment, it can be not valid in another one* “ ([Zuse98], S. 409f). Die Beschäftigung mit der summativen Validierung ist ein mächtiges Hilfsmittel zur Lösung des Maß-Selektions-Problems (vgl. Kapitel 3.6.2), da während der Validierung typische Charakteristika des untersuchten Maßes offenkundig werden, die Aufschluß über dessen jeweils spezifische Gebrauchstauglichkeit geben können.

- *Formative Validierung*: Während der Erstellung eines neuen Maßes soll explizit auf die Forderungen der Meßtheorie eingegangen werden, um bereits frühzeitig potentielle Schwächen des Maßes erkennen zu können. Bei konsequenter Einhaltung der in den vorigen Abschnitten aufgezeigten Forderungen bzw. des in Abschnitt 4.4 formulierten Prozesses findet die formative Validierung folglich implizit statt.

Für die interne Validierung werden häufig sogenannte *Anforderungskataloge* angegeben, in denen Anforderungen an die zu validierende Skala angegeben sind. In vielen Fällen sind diese Anforderungen allerdings zu restriktiv, so daß nicht jede Skalenart unterstützt wird. Eine weitere Schwäche ist die Unklarheit, inwieweit diese Anforderungskataloge notwendige und/oder hinreichende Kriterien zur Validierung darstellen. Aufgrund der Vielzahl existierender Anforderungskataloge soll exemplarisch nur der bekannteste vorgestellt und diskutiert werden. Im Anschluß wird ein kurzer Überblick über weitere Anforderungskataloge gegeben, da diese häufig ursächlich an der Entstehung des Meßtheorie-Anwendungs-Problems mitwirken (vgl. Kapitel 3.6.2).
Der wohl bekannteste Anforderungskatalog ist der von Weyuker [Weyu88]: Er umfaßt die folgenden 9 Punkte, die sich auf das ERS, das FRS und auf das dazwischen definierte Maß beziehen:

1. *Unterschiedlichkeit der Meßwerte*: Es muß mindestens zwei Entitäten geben, deren Meßwerte unterschiedlich sind.
2. *Endlichkeit der Entitäten*: Für eine nicht-negative Zahl c als Meßwert gibt es lediglich eine endliche Anzahl von Entitäten mit dem Meßwert c (negative Meßwerte werden hier bereits ausgeschlossen).
3. *Äquivalenzklassen*: Es gibt unterschiedliche Entitäten, die auf denselben Meßwert abgebildet werden.
4. *Aufgabenunabhängigkeit*: Es gibt unterschiedliche Entitäten, die jeweils dieselbe Aufgabe erledigen (z.B. dieselbe Funktionalität besitzen), aber auf unterschiedliche Meßwerte abgebildet werden.
5. *Schwache Ordnung bzgl. Verkettung*: Für beliebige Entitäten p, q gilt, daß die Verkettung von $p \circ q$ einen größeren (oder gleich großen) Meßwert liefert als von p und q jeweils alleine, d.h. $\mu(p \circ q) \geq \mu(p) \wedge \mu(p \circ q) \geq \mu(q)$.
6. *Unterschiedlichkeit durch Verkettung*: Für beliebige, unterschiedliche Entitäten p, q mit denselben Meßwerten gibt es eine weitere Entität r, so daß jeweils die Meßwerte der Verkettungen von $(p \circ r)$ zu $(q \circ r)$ und von $(r \circ p)$ zu $(r \circ q)$ unterschiedlich sind, d.h.
   $\forall p, q, \mu(p) = \mu(q): \exists r : \mu(p \circ r) \neq \mu(q \circ r) \wedge \mu(r \circ p) \neq \mu(r \circ q)$.
7. *Unterschiedlichkeit durch Reihenfolge*: Für eine gegebene Entität p ist eine Entität q durch Änderung der Reihenfolge der in p enthaltenen Objekte (z.B. Anweisungen) erstellbar, so daß p und q jeweils unterschiedliche Meßwerte besitzen.
8. *Namensunabhängigkeit*: Für beliebige Entitäten p, q gilt, daß wenn q lediglich durch Umbenennung von p entstanden ist, p und q dieselben Meßwerte besitzen.
9. *Summe-Teil-Verhältnis*: Das ganze muß mindestens einen so großen Meßwert besitzen wie die Summe der Meßwerte der Teile.

Diese Anforderungen, die z.B. in [ChKe94] und [AlKh95] zur internen Validierung der jeweils dort vorgestellten Softwaremaße verwendet werden, besitzen einige Nachteile, die typisch für diese Art von formalen Kriterien sind:

- Der Nachweis, daß ein Maß den Weyuker-Anforderungskatalog erfüllt, ist *nicht hinreichend*: So wird z.B. in [ChSm91] ein künstliches, bewußt und offensichtlich sinnloses Maß vorgestellt, das trotzdem allen 9 Anforderungen genügt.

- Der Nachweis, daß ein Maß den Weyuker-Anforderungskatalog erfüllt, ist *nicht notwendig*. Ein sinnvolles, intern valides Messen ist innerhalb der Nominalskala möglich, obwohl dort z.B. niemals die Bedingung 5 erfüllt sein kann, da innerhalb der Nominalskala keine Ordnungsrelation definiert ist. Die zu restriktive interne Validierung durch den Weyuker-Anforderungskatalog wird in [KiPfFe95] diskutiert.

Die Möglichkeit, trotz dieser Vorgaben nicht intern valide Maße zu erstellen, macht ein Problem deutlich: Die mangelnde Beschäftigung mit der Ausgangsbasis jeden Messens, dem empirischen Relationensystem. Statt dort auf die explizite Benennung der konsensfähigen Relationen auf konsensfähigen Produktmodellen zu setzen, werden formale Kriterien aufgezeigt, die größtenteils auf die Abbildung in das FRS abzielen. Eine interne Validierung kann dadurch zwar unterstützt, jedoch nicht garantiert werden. Ähnliche, eher auf das FRS abzielende Anforderungskataloge sind z.B. die von Lakshmanan, Jayaprakash und Sinja [LaJaSi91] (für die Maßvalidierung verwendet von Allen und Khoshgoftaar, vgl. [Zuse98], S. 391ff) und Bache, der sie darüber hinaus auch für die Validierung eigener Maße (die *VINAP*-Maße) verwendet hat (vgl. [FePf96], S. 332).

Neben diesen auf das FRS fokussierten, eher formal formulierten und teilweise sehr restriktiven Anforderungskatalogen existieren weitere, eher umgangssprachlich formulierte und allgemeinere Anforderungskataloge. Diese haben den Nachteil, daß sie aufgrund ihrer Allgemeinheit und ihrer umgangssprachlichen Formulierung erst noch für eine interne Validierung operationalisiert werden müssen. Die nach einer Operationalisierung identifizierbaren Anforderungen sind allerdings dann i.d.R. implizit bei konsequenter Anwendung des in Abschnitt 4.4. formulierten Prozesses erfüllt.
Exemplarisch für diese allgemeinen Anforderungskataloge, die häufig auch als Axiome formuliert werden, soll hier der von Shepperd vorgestellte Katalog beschrieben werden (vgl. [Zuse98], S. 366ff):

1. Alle Parameter, die Einfluß auf den Meßprozeß nehmen können, müssen eindeutig spezifiziert sein, so daß die Messung wiederholbar wird.
2. Die Messung muß wenigstens zwei verschiedene Äquivalenzklassen aufspannen.
3. Die Gleichheitsrelation muß sinnvoll definiert sein.
4. Die Entitätsmenge darf nicht unendlich groß sein, da anderenfalls unterschiedliche Entitäten des ERS auf dieselben Entitäten des FRS abgebildet werden können, was die Interpretation der Meßwerte erschwert.
5. Das Maß darf keine Anomalien erzeugen, d.h. es muß z.B. die empirischen Relationen innerhalb des ERS erhalten.
6. Die Aussagen, die auf dem FRS vorgenommen werden, müssen invariant gegenüber zulässigen Transformationen innerhalb des FRS sein.

Nach einer Operationalisierung dieser von Shepperd aufgestellten Anforderungen (z.B. welche Parameter werden durch Forderung 1 angesprochen oder welche Art von Anomalien in Forderung 5 müssen vermieden werden) ergeben sich Anforderungen, die bei konsequenter Einhaltung der im vorigen Abschnitt aufgezeigten Forderungen bzw. des in Abschnitt 4.4 formulierten Prozesses automatisch erfüllt sind. Lediglich bzgl. der Praktikabilität der Messung werden einige zusätzliche Kriterien genannt: So ist das Abbilden aller Entitäten auf ein und dieselbe Entität des FRS zwar möglich, praktisch aber wertlos, d.h. die Forderung 2 wird lediglich von theoretisch konstruierbaren Extremmaßen nicht eingehalten. Die Forderung 4 belegt nur, daß bei einem FRS, bei dem wenigen Entitäten eine Vielzahl von unterschiedlichen Entitäten des ERS zugewiesen werden, die Aussagekraft gering ist, da auch das ERS nicht genügend Unterscheidungen vornimmt.

Ähnliche, jeweils zu interpretierende und allgemein formulierte Anforderungskataloge werden z.B. angeboten von Basili und Reiter [BaRe79] und Jones [Jone94].
Die gezeigten Schwächen bzgl. der Anforderungskataloge für eine interne Validierung vermindern nicht deren Notwendigkeit. Die formative interne Validierung wird daher implizit innerhalb der Anwendung des in Abschnitt 4.4. vorgestellten Prozesses zur Herleitung meßtheoretisch konformer Skalen vorgenommen. Dieser Prozeß unterstützt die gesamte Bandbreite des Vermessens, d.h. das Messen innerhalb einer Nominalskala bis hin zum Messen innerhalb der Absolutskala.
Die summative interne Validierung zur Verwendung existierender Maße (z.B. aus der Literatur) oder zur kontextbezogenen Überprüfung verläuft ähnlich und wird ebenfalls in Abschnitt 4.4. behandelt.

4.3.2 Externe Validierung

Bereits aus der Definition der externen Validierung wird deutlich, daß sie nicht unabhängig von der internen Validierung betrachtet werden kann, sie baut vielmehr auf ihr auf. Die Aufgabe der externen Validierung ist also, die Eigenschaften der intern validen Skala praktisch relevant auf Fragestellungen in der realen Welt zu übertragen. Da diese Fragestellungen durch die Erstellung eines angepaßten Qualitätsmodells (vgl. Kapitel 2) konkretisiert sein sollten, ist das Ziel der externen Validierung die statistische Überprüfung der Korrelation der Meßergebnisse mit übergeordneten Qualitätsmerkmalen innerhalb des dazugehörigen Qualitätsmodells. Daraus folgt, daß diese Form der Validierung entlang statistischer Überprüfungen deutlich unzuverlässiger ausfällt, da hier die Korrelation zwischen internen und externen Merkmalen versucht wird: „*Given our poor understanding of the relationships between various software products and processes, external validation seems highly unreliable*“ ([Baker et al.90a], S. 280).
Eine wesentliche Forderung der externen Validierung ist die im Vorfeld zu erstellende Hypothese, die anschließend mittels statistischer Techniken validiert wird [Fent94]. Die Hypothese muß auf einem kausalen Begründungsmuster beruhen, d.h. die Hypothese muß einen begründeten Verdacht darlegen, warum das Maß Einfluß auf das Qualitätsmerkmal haben soll.
Werden diese Vorbedingungen der externen Validierung nicht eingehalten, so droht die Gefahr der „*Shotgun-Correlations*“ [CoGu93]. Dieser falsche Ansatz besteht aus den folgenden vier Schritten:

1) Sammlung von möglichst vielen Daten: Dies umfaßt verschiedenartigste Meßdaten als auch empirisch gewonnene, historische Daten bzgl. des Qualitätsmerkmals (z.B. Wartungskosten).
2) Berechnung der Korrelation von möglichst vielen Meßwerten gegenüber den empirisch gewonnenen Daten.
3) Identifikation einer besonders guten Korrelation.
4) Identifikation einer Hypothese.

Bei entsprechend großem Datenvolumen und dem Einsatz beliebig komplexer Polynome kann die Identifikation einer besonders guten Korrelation automatisch erfolgen. So wird z.B. in [OmHa92] ein Prozeß vorgestellt, der, aufbauend auf den empirischen Daten des Qualitätsmerkmals Wartbarkeit für 8 Projekte innerhalb der Firma *Hewlett-Packard*, 60 Maße anwendet, mittels der Hauptkomponentenanalyse (vgl. Kapitel 6.4.2) Redundanzen entfernt und anschließend eine minimale Menge der berechneten Maße bestimmt, die bzgl. der Vermessung von Wartbarkeit nützlich sind [ebd. S. 1]. Ein Ergebnis der Anwendung dieser Methode ist die „*4 metric MI equation*“, ein Softwaremaß, das das Qualitätsmerkmal Wartbarkeit bestimmen soll:

Wartbarkeit = 171
- 3.42 * ln (durchschnittlicher *HalsteadEffort*-Wert)
- 0.23 * (durchschnittliche zyklomatische Zahl)
- 16.2 * ln (durchschnittlicher LOC-Wert)
+ 0.99 * (durchschnittliche Anzahl von LOC pro Kommentar)

In [OmWe95] wird dieses Maß noch weiter angepaßt und aufbauend auf der Korrelation mit den zur Verfügung stehenden empirischen Daten folgende Klassifizierung vorgenommen:

- Softwaremodule mit einem Wert größer 85 sind sehr gut wartbar.
- Softwaremodule mit einem Wert zwischen 65 und 85 sind mit moderatem Aufwand wartbar.
- Softwaremodule mit einem Wert kleiner 65 sind nur schwer wartbar.

Derartig fragliche Ansätze werden auch heute noch angewendet (z.B. wird in [FiNe01] nach Anwendung von 226 Maßen auf acht Programme, zu denen Daten bzgl. der Fehleranfälligkeit vorlagen, und einer wiederum auf der Hauptkomponentenanalyse basierenden Selektion von 42 Maßen zur Erreichung einer statistisch signifikanten Korrelation zum externen Qualitätsmerkmal Fehleranfälligkeit ein derartiges Polynom erstellt) und können nur durch konsequente Anwendung der internen und externen Validierung, wie sie in dieser Arbeit vorgeschlagen werden, falsifiziert werden. So attraktiv solche hybriden Maße für die Bestimmung von so relevanten Qualitätsmerkmalen wie Wartbarkeit auch sind, so fatal wären Entscheidungen, die lediglich auf einer solchen Korrelation beruhen. Mittels derselben Technik der externen Validierung läßt sich beispielsweise eine signifikante Korrelation zwischen der Storchpopulation in Skandinavien und der Bevölkerungspopulation zeigen [KiPfFe95]. In [CoGu93] wird ein computersimuliertes Experiment vorgestellt, in dem signifikante Korrelationen zwischen zufällig erzeugten Werten untereinander und zu einigen Halstead-Maßen identifiziert werden.
Wegbereiter der externen Validierung, die aufbauend auf einer internen Validierung kausal begründete Hypothesen einer Korrelation zu Qualitätsmerkmalen aufzuzeigen versucht, sind Arbeiten von Schneidewind [Schn92], die größtenteils in die IEEE1061 eingeflossen sind und dort als *Richtlinie zur Validierung von Softwaremaßen* bezeichnet werden (vgl. [Zuse98], S. 352ff).
Wesentliches Grundgerüst dieser externen Validierungstechnik ist das Aufzeigen von Abhängigkeiten der einzelnen Validierungsschritte, die sich auf eine zeitliche Abfolge projizieren lassen. Die Schritte lassen sich mit Hilfe eines Aktivitätendiagramms (vgl. z.B. [FoSc98]) wie in Abbildung 16 dargestellt verdeutlichen:

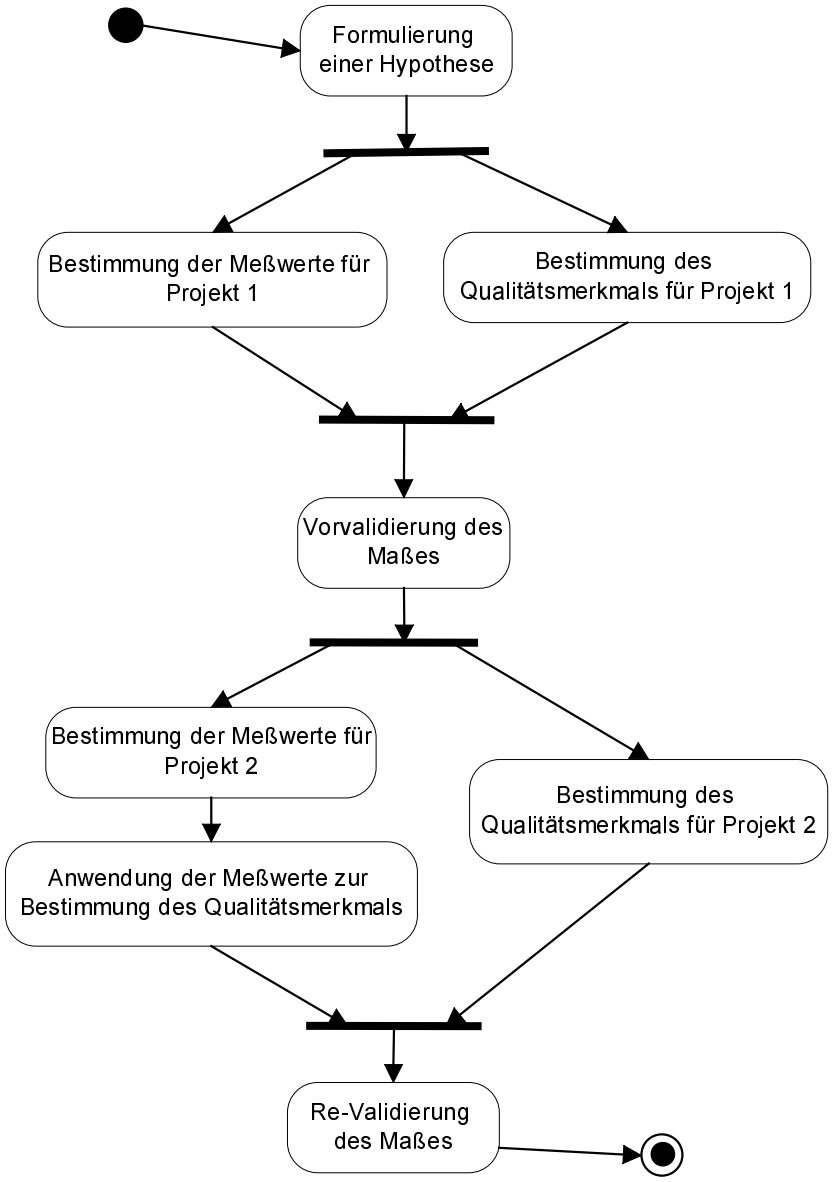


Abbildung 16: Prozeß zur externen Validierung von Softwaremaßen, nach [Schn92], S. 410f[13]

Der Prozeß läßt sich grob in die zwei folgenden Teilprozesse aufteilen:

1. *Vorvalidierung des Maßes*, d.h. die manuelle Überprüfung der Hypothese, die eine Korrelation zwischen Maß und Qualitätsmerkmal vermutet. Dabei werden für ein Validierungsprojekt $\mathcal{P}_1$ sowohl die Meßdaten als auch das Qualitätsmerkmal selbst (z.B. in Form von Entwicklerbefragungen) ermittelt. Das Ergebnis ist ein potentiell valides Maß zur Bestimmung des Qualitätsmerkmals, d.h. es gilt zumindest für Projekt $\mathcal{P}_1$ das *Validitätsprinzip* (dort: „*principle of validity*“, [Schn92], S. 412), das für die bezüglich des Maßes sinnvollen

[13] Schneidewind fordert im Gegensatz zum hier dargestellten Prozeß die zeitliche Hintereinanderausführung der beiden Aktivitäten „Meßwertbestimmung“ und „Qualitätsmerkmalsbestimmung“ (sowohl innerhalb der Vorvalidierung als auch innerhalb der Re-Validierung). Wichtig ist dabei nur, daß die Bestimmung der Meßwerte unabhängig von der Bestimmung der Qualitätsmerkmale geschieht. Bei entsprechender Disziplin oder bei der Möglichkeit des Outsourcings einer der beiden Aktivitäten können beide Aufgaben aber durchaus parallel stattfinden.

Relationen besagt, daß Meßwerte genau dann in Relation stehen, wenn die Qualitätsmerkmale in Relation stehen.

2. *Re-Validierung des Maßes*, d.h. die Überprüfung ob die Anwendung des potentiell validen Maßes ebenfalls sinnvoll für ein Anwendungsprojekt $\mathcal{P}_2$ angewendet werden kann, d.h. ob auch für $\mathcal{P}_2$ das Validitätsprinzip gilt. Das Ergebnis ist die Validierung (oder Falsifizierung) des Maßes bzgl. des in der Hypothese vorgeschlagenen Qualitätsmerkmals.

Das Validitätsprinzip verlangt nach einer im Vorfeld getätigten internen Validierung, da ansonsten die für ein Maß sinnvollen Relationen noch unklar sind. Polynome beliebiger Komplexität scheitern häufig alleine an dieser Forderung: „*Researchers try squares, logarithms, exponentials and other functions or combinations to obtain the best fit possible, without having theoretical reasons for such manipulations*“ ([CoGu93], S. 5). Wird ein Maß beispielsweise innerhalb einer ordinalen Skala verwendet, so ist die Verwendung dieses Maßes als Rationalskala für die Bestimmung eines Qualitätsmerkmals nicht geeignet.
Das grundsätzliche Vorgehen einer externen Validierung entlang dieses Prozesses birgt folgende Risiken, sogenannte *fundamentale Probleme der Maßvalidierung* ([Schn92], S. 416), die beachtet werden müssen: Große Zeitspannen, relevante Produkt- oder Prozeßunterschiede und sich verändernde Ziele und Umgebungen während der Validierung erhöhen die Anzahl der veränderlichen Variablen, so daß die Validitätsüberprüfung bzgl. des Qualitätsmerkmals verfälscht werden kann. So kann z.B. eine Veränderung der Ziele zu unterschiedlichen Ergebnissen der direkten Bestimmung des Qualitätsmerkmals von Projekt $\mathcal{P}_1$ und Projekt $\mathcal{P}_2$ führen. Da diese Veränderungen nicht durch die Maße nachgeführt werden, wird die Re-Validierung wahrscheinlich die Falsifizierung der Hypothese zur Folge haben. Ebenfalls möglich sind Veränderungen des Produkts innerhalb eines Projekts selbst (sowohl im ersten als auch im zweiten Teilprozeß), so daß sich die Meßwerte und die Bestimmung der Qualitätsmerkmale auf unterschiedliche Produkte beziehen.

Da die externe Validierung entsprechend dieses Prozesses mit viel Aufwand bzw. bzgl. der fundamentalen Probleme der Maßvalidierung je nach betrachtetem Projekt nur unter Tolerierung von Kompromissen möglich ist, sind erst wenige Arbeiten entsprechend dieser Vorgaben getätigt worden. Der Teilprozeß „Vor-Validierung“ ist z.B. in [BrMoBa98] entlang obigen Vorgehens durchgeführt worden. Folgende Motivation begründet dort einige explizit formulierte und vor-validierte Hypothesen: „*Our goal was to define and validate a set of high-level design measures to evaluate the quality of the high-level design of a software system with respect to its fault-proneness [...]*“([ebd.], S. 2). Die Hypothesen umfassen Vermutungen über die Abhängigkeit von Fehler-anfälligkeit zur Kohäsion (Hypothese *H-CH*), zur Kopplung (Hypothese *H-IC*) und zur Größe (Hypothese *H-A*). Die Re-Validierung mit weiteren Projekten wird dort allerdings nicht vorgenommen, sondern als Aussicht für weitere Forschungsarbeit im Papier vorgeschlagen.

Die beiden wesentlichen Probleme der externen Validierung sind:

- Mangelnde intern valide Maße, auf der eine externe Validierung aufbauen kann, und
- mangelnder Konsens bzgl. allgemeiner Qualitätsmerkmale, gegenüber denen dann die externe Validierung vorgenommen werden soll. So wird z.B. in [GiSe89] berichtet, wie sehr die Meinungen von Wartungsentwicklern bzgl. der Einschätzung der Wartbarkeit eines einzelnen Produkts auseinander liegen (die Einschätzungen reichen gleichmäßig von „*sehr einfach wartbar*“ bis „*sehr schwer wartbar*“) und wie sehr diese teilweise objektiv feststellbaren Daten (z.B. in Form von gemessenen tatsächlichen Wartungskosten) widersprechen.

Diese Arbeit umfaßt im folgenden beide Arten der Validierung von Maßen. Zuerst wird die interne Validierung vorgenommen (vgl. Kapitel 6), die schließlich als Vorbedingung einer darauf

aufbauenden externen Validierung benötigt wird. Die darauf aufbauenden Hypothesen, die bzgl. der jeweiligen Prozesse, in denen Maße angewendet werden sollen, definiert werden (vgl. Kapitel 8 und Kapitel 9), sind jeweils durch umfangreiche empirische Arbeiten sowohl vor-validiert, als auch in Folgeprojekten re-validiert worden. Die Möglichkeit der externen Validierung wird in dieser Arbeit ermöglicht, indem der nötige Konsens durch detaillierte Qualitätsmodelle (vgl. Kapitel 2) als auch durch explizite Schritte während der Erstellung von Maßen hergestellt wird. Dabei konzentriert sich diese Arbeit auf eine Klasse von Maßen, die in der Lage sind, Kohäsion zu bestimmen (vgl. Kapitel 7). Neben der Vorstellung von validierten Maßen dieser Klasse steht folglich der Prozeß zur Erstellung aufgabenangepaßter und validierter Maße im Vordergrund (vgl. Kapitel 7.3.1 und 7.3.2).
Im folgenden wird für die interne Validierung ein Prozeß zur Erstellung von Maßen allgemein vorgestellt, indem die interne Validierung implizit vorgenommen wird. Damit können erstmals viele der in Kapitel 3.6.2 aufgezeigten Probleme reduziert werden, was im Kapitel 8 und 9 mit Beispielen aus Projekten, die im Rahmen dieser Arbeit durchgeführt wurden, praktisch belegt wird. Ein Fokus dieser Arbeit, die Bestimmung von Kohäsion, führt in Kapitel 7.3 zu einer Spezialisierung dieses Prozesses für die Erstellung meßtheoretisch konformer Skalendefinitionen von Kohäsionsmaßen.

4.4 Prozeß zur meßtheoretisch konformen Skalendefinition

Ausgehend von den in diesem Kapitel gemachten Anforderungen an ein meßtheoretisch konformes Vermessen von Softwareprodukten kann ein Vorgehensmodell definiert werden, das alle Schritte von der Festlegung des Maßeinsatzes und des Qualitätsmodells, über die Merkmalsidentifizierung bis hin zur Extraktion der auf der Meßwertinterpretation basierenden Resultate enthält (vgl. Abbildung 17). Die neun jeweils aufeinander aufbauenden Schritte bedeuten dabei im einzelnen:

1. Jeder Vermessung liegt die problembasierte Festlegung des Maßeinsatzes (vgl. Kapitel 3.3) und die Definition eines angepaßten Qualitätsmodells zugrunde (vgl. Kapitel 2). Die Einflußfaktoren auf das Qualitätsmodell sind hier ebenso wenig modelliert wie die eventuellen Abhängigkeiten zwischen Maßeinsatz und Qualitätsmodell.
2. Im nächsten Schritt wird eine zu betrachtende Eigenschaft soweit verfeinert, bis sie als Merkmal bezeichnet werden kann (vgl. Abschnitt 4.1). Dieser Verfeinerungsschritt kann eventuell bereits Bestandteil der Qualitätsmodelldefinition sein (vgl. Kapitel 2). Da ein Qualitätsmodell in der Regel mehrere zu verfeinernde Eigenschaften aufweist, wird dieses Vorgehensmodell für jede Eigenschaft separat instanziiert.
3. Anschließend wird eine Abstraktionsfunktion definiert, die das betrachtete Merkmal bestmöglich darstellt und so die Betrachtung darauf fokussiert (vgl. Abschnitt 4.1.1). Die Konkretisierung der Abstraktionsfunktion im Falle ihrer Unvollständigkeit, Mehrdeutigkeit, Inkonsistenz oder Unverständlichkeit (vgl. Abschnitt 4.1.1) kann speziell bei dem Versuch der Wiederverwendung von Maßen und deren dafür notwendige Rekonstruktion der Produktmodellerstellung schwierig sein und kontextabhängige Vorgaben notwendig machen.
4. Daraufhin werden auf dem Ergebnis der Abstraktionsfunktion, d.h. auf Produktmodellen von Softwareprodukten, bzgl. des betrachteten Merkmals Relationen identifiziert. Bereits hier sollte überprüft werden, inwieweit die Relationen auf dem Produktmodell auf die zugrunde liegenden Produkte übertragen werden können.
   Sollte über die vorgeschlagenen Relationen kein Konsens innerhalb des Anwendungsbereichs der Vermessung möglich sein, so kann die Aussagekraft der Relationen limitiert werden (z.B. durch Beschränkung eines als verhältnisskaliert vorgestellten Merkmals auf die Ebene eines ordinalskalierten Merkmals), sie kann diskutiert werden und eventuell sogar als kontextspezifische Setzung manifestiert werden (z.B. als Teil der Qualitätsanforderungen bei einer Auftragsarbeit).

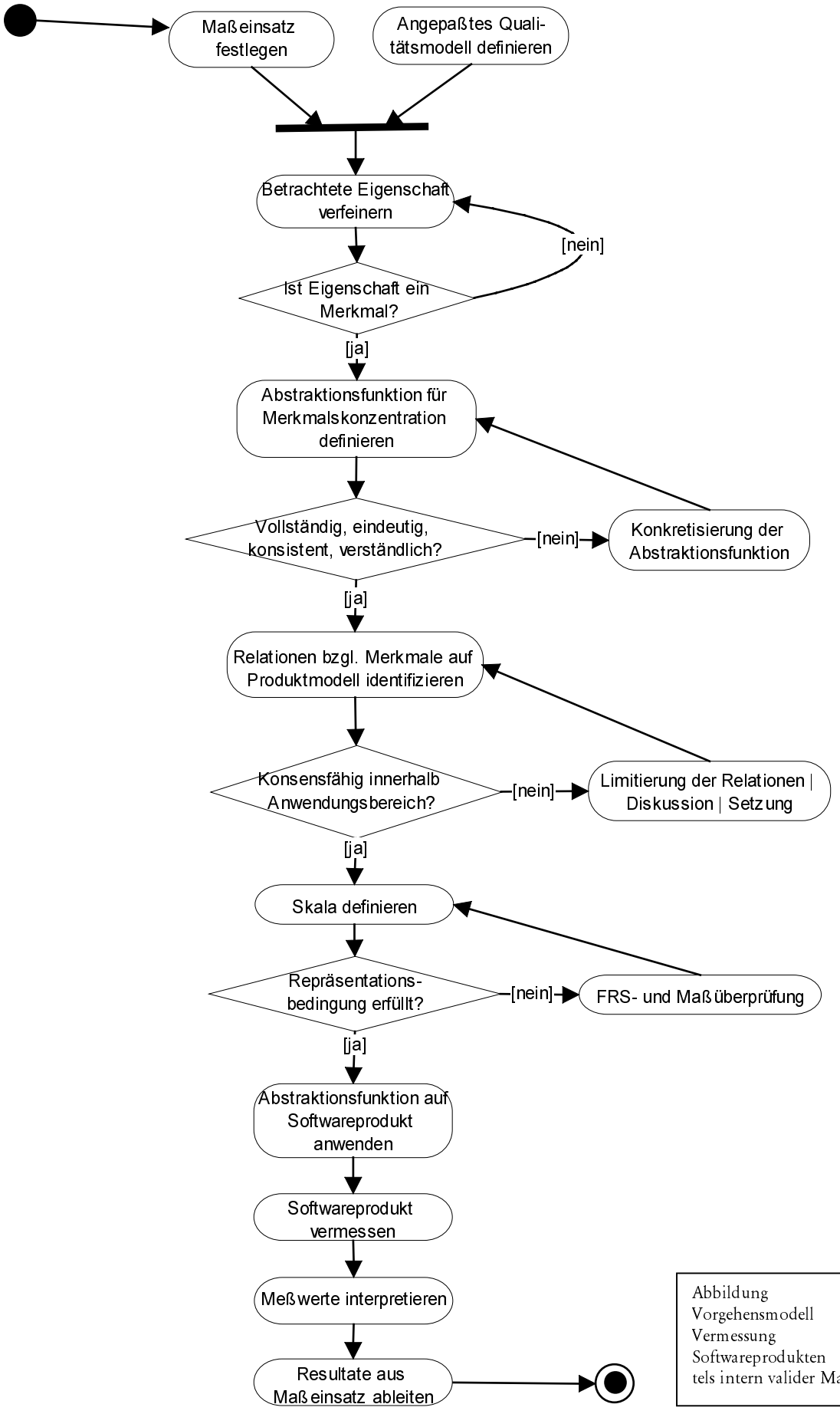


Abbildung 17: Vorgehensmodell zur Vermessung von Softwareprodukten mittels intern valider Maße

5. Auf Grundlage des durch die vorherigen Schritte definierten empirischen Relationensystems kann ein formales Relationensystem und ein dazwischen abbildendes, relationenerhaltendes Maß formuliert werden. Diese Teilaufgabe ist i.d.R. nicht schwierig, da $\mathbb{R}$ für alle bekannten Maße genügt und auch für neu erstellte Maße genügend unterscheidbare Elemente und Relationen zur Verfügung stellt (so sind z.B. alle in [LoKi94] definierten Maße Abbildungen auf $\mathbb{R}$). Die Repräsentationsbedingung ist folglich einfach durch den Test überprüfbar, ob die beiden Relationensysteme und das dazwischen abbildende Maß wirklich eine Skala bilden.
   Alle bis zu diesem Schritt anfallenden Zwischenergebnisse können für spätere Messungen wiederverwendet werden. Ein deutlich veränderter Kontext (z.B. geänderte Programmiersprache, andere Softwareentwickler oder auch ein neues CASE-tool) kann dabei allerdings Modifikationen notwendig machen (z.B. Anpassung der Abstraktionsfunktion an die neue Programmiersprache, Änderungen der Relationen für einen Konsens mit den anderen Entwicklern oder ein geändertes Qualitätsmodell mit dadurch veränderten Merkmalen/Eigenschaften, um den Schwächen und Stärken eines neuen CASE-tools gerecht zu werden).

Die Anwendung des Maßes geschieht in den folgenden vier Schritten (vgl. Abbildung 15 in Abschnitt 4.3)

6. Die Abstraktionsfunktion wird auf das zu betrachtende Softwareprodukt angewendet. Das Ergebnis ist ein konkretes Produktmodell.
7. Die Vermessung geschieht durch Anwendung des Maßes auf das Produktmodell.
8. Die Interpretation der durch die Vermessung erzeugten Meßwerte geschieht unter Verwendung der innerhalb des verwendeten Skalentyps zulässigen statistischen Techniken (vgl. Abschnitt 4.2.1 – Abschnitt 4.2.5). Die Interpretation geschieht innerhalb des Produktmodells.
9. Die Rückübertragung der Ergebnisse innerhalb des formalen Relationensystems auf das Produktmodell führt zu Resultaten, die dem Zweck der zu Beginn festgelegten Parameter (Qualitätsmodell und Maßeinsatz) dienlich sind und die sich daher auf die das Produktmodell bildenden Elemente beziehen. Die Rückübertragung muß dabei eventuelle Einschränkungen der Rückübertragung von Elementen des ERS auf die vermessenen Produkte berücksichtigen (vgl. Punkt 4).

Die im letzten Punkt formulierte Dienlichkeit der Resultate verlangt allerdings noch nach einer externen Validierung (vgl. Abschnitt 4.3.2); die interne Validierung wird in diesem Prozeß explizit mit vorgenommen. Damit dieser Prozeß auch praxisrelevant angewendet werden kann, sollte bei den Punkten 3 (Abstraktionsfunktion definieren), 5 (Skalendefinition), 6 (Produktmodell-erstellung), 7 (Vermessung) und teilweise 8 (Interpretation; hier besonders: statistische Techniken) auf eine mögliche Werkzeugunterstützung geachtet werden. Fixe Softwareprodukt-Meßwerkzeuge, d.h. bzgl. des Qualitätsmodells und der anzuwendenden Maße nicht konfigurierbare Programme passen unter Umständen bereits zum aktuellen Kontext. Ist dies nicht der Fall, sollte auf alternative Werkzeuge (z.B. vielfältig Anpaßbare wie Crocodile) ausgewichen werden. In diesen können einige Teilschritte des Prozesses direkt abgebildet werden.
Bei der Wiederverwendung von Maßen ist es hilfreich, ebenfalls diesen Prozeß zu durchlaufen und lediglich Aktivitäten, die durch die Wiederverwendung bereits abgedeckt sind, zu überspringen. Allerdings sind speziell die Schritte 3 (Abstraktionsfunktion definieren) und 4 (Relationenfindung) fast nie explizit gemacht. Dies ist aber absolut notwendig, um vielen Problemen bei der Anwendung von Softwaremaßen (vgl. Kapitel 3.6) zu begegnen. Werden die einem Maß zugrundeliegenden Relationen beispielsweise nicht im Anwendungskontext überprüft, besteht die Gefahr, daß die Meßergebnisse für den aktuellen Kontext wertlos sind, da die Repräsentationsbedingung nicht erfüllt ist (schließlich stimmen in einem solchen Fall bereits die Relationen und die darauf aufbauenden Aussagen im ERS nicht).

4.5 Zusammenfassung

Die Anwendung der Meßtheorie erlaubt sowohl das begründbar korrekte, zielorientierte Vermessen als auch die Identifikation von falschen Maßen und deren ebenfalls unsinnigen Resultaten. Grundlage sind dabei Relationensysteme, die aus Objekten eines bzgl. eines Merkmals abstrahierten Produktmodells und konsensfähigen Relationen darauf bestehen. Bereits hier weisen viele aus der Literatur bekannte Maße große Defizite auf. Die Menge möglicher Relationen korrespondiert dabei unmittelbar mit der Aussagekraft der graduellen Ausprägungen der betrachteten Merkmale. Die Aussagekraft dieses ERS kann während des gesamten Messens nicht übertroffen werden, d.h. Aussagen, die innerhalb des formulierten ERS nicht möglich sind, werden es auch später während der Vermessung in der Praxis nicht sein. Für die Vermessung von Softwareprodukten ist eine wesentliche Voraussetzung für ihre Überführung in ein Produktmodell die vollständige, eindeutige, konsistente und verständliche Abstraktionsfunktion. Ausgehend vom Merkmalsniveau kann unter Verwendung eines formalen Relationensystems und einem Maß eine Skala mit maximal demselben Niveau erstellt werden. Die Überprüfung, ob die Skala, die auf dem FRS vorgenommenen statistischen Techniken und die darauf beruhende Interpretation meßtheoretisch konform stattfinden, wird als interne Validierung bezeichnet; die Überprüfung des Zusammenhangs der Ergebnisse zu relevanten, übergeordneten Eigenschaften und den damit verbundenen Fragestellungen wird als externe Validierung bezeichnet.
Die in der Meßtheorie begründeten Anforderungen ermöglichen die Definition eines in der Praxis eingesetzten Prozesses zur Vermessung von Softwareprodukten mittels intern valider Maße. Seine Anwendung ermöglicht den direkten Einstieg in den ebenfalls vorgestellten Prozeß zur externen Validierung. Damit ist ein konstruktiver Weg aufgezeigt, wie die Anwendung der Meßtheorie viele der im Bereich der Softwareproduktvermessung auftretenden Probleme beseitigen helfen kann.

„Bevor wir verstehen,
was die Erweiterung einer Sache ist,
müssen wir verstehen,
was die Sache eigentlich ist“
(Superjazzer Wynton Marsalis im
Feuilleton der Welt vom 26.5.2000)

5 Graphenmodell bisheriger Maße

Mit der konsequenten Anwendung des im letzten Kapitel vorgestellten Prozesses zur meßtheoretisch konformen Skalendefinition werden viele der in Kapitel 3.6.2 aufgeführten, spezifischen Probleme beim Einsatz von Produktmaßen behoben:

- Die verwendeten Maße sind nach Durchlaufen des Prozesses exakt definiert, das zugrundeliegende Produktmodell ist vollständig, eindeutig, konsistent und verständlich beschrieben (vgl. Maß-Definitions-Problem).
- Die verwendeten Maße besitzen aufgrund der im Prozeß vorgeschriebenen Vorgehensweise notwendigerweise ein Pendant in der realen Welt, da sie erst nach Etablierung von konkreten Entitäten und aussagekräftigen Relationen innerhalb des ERS betrachtet werden (vgl. Wert-Interpretations-Problem).
- Die vorgenommene Limitierung von Maßen auf das Bestimmen von gegenwärtig existierenden Merkmalen ermöglicht die Vorhersage mittels Maßen nur noch durch das Aufzeigen einer kausalen Verbindung zwischen dem betrachteten Merkmal und davon abhängigen, eventuell in die Zukunft gerichteten Qualitätseigenschaften. Die Vorhersage geschieht folglich nicht mehr direkt durch das Maß, sondern wird durch die Definition des Qualitätsmodells auf die Abhängigkeit zwischen zukünftigen Eigenschaften und gegenwärtigen Merkmalen verfeinert (vgl. Messen-Vorhersage-Problem).
- Existierende Maße, die nicht intern validierbar sind, werden nicht mehr zugelassen. Die damit verbundenen Probleme, das Anforderungsdenken des Managements an Softwaremaße auf das sinnvoll Machbare zu reduzieren, stellen ein allgemeines Problem bei der Einführung neuer Techniken innerhalb des Software-Engineering dar (vgl. besonders die Problemfelder „Vorgehen“, „Management“ und „Ausbildung und Unterstützung“ in Kapitel 3.6.1 und das Qualitätsmodell-Problem).
- Das vorgeschlagene Vorgehensmodell verbindet erstmalig die Anforderungen der Meßtheorie mit denen der Meßanwender. Dies wird durch eine Konzentration auf die Elemente des ERS erreicht (vgl. Meßtheorie-Anwendungs-Problem).

Dieser Abschnitt versucht nun, aufbauend auf den Ergebnissen des letzten Kapitels die Vielzahl von existierenden Softwaremaßen und die dafür jeweils notwendigen Produktmodelle einheitlich mittels eines Graphenmodells zu beschreiben. Anhand dieses allgemeinen Graphenmodells ist es einfacher, die existierenden Maße (und deren jeweils notwendigen Produktmodelle) zu klassifizieren (vgl. Maß-Selektions-Problem), Probleme der Rückprojektion der Meßwerte auf die ihnen entsprechenden Softwareteile zu konkretisieren und diesbezügliche Lösungen vorzustellen (vgl. Wert-Rückprojektions-Problem) sowie weitere Möglichkeiten der Visualisierung von Meßwerten herzuleiten (vgl. Multiwert-Problem). Ein wesentlicher Schritt dafür ist die Identifikation einer wichtigen Erweiterung bisheriger Maßansätze, der in den weiteren Abschnitten dieser Arbeit nachgegangen wird, da hiermit ein theoretisch fundierter, bzgl. der betrachteten Systemgrößen skalierender und in mehreren im Rahmen dieser Arbeit durchgeführten Projekten praktisch validierter Ansatz für die Softwareproduktvermessung gegeben ist.

5.1 Messen als Knotenattributierung

Grundlage jedes Meßvorgangs sind Produktmodelle (vgl. Kapitel 4.1.1). Jedes System, das vermessen werden soll, muß zuvor entsprechend der gewählten Abstraktionsfunktion modelliert werden. Ein Produkt kann dabei in verschiedene Produktmodelle überführt werden. Eine Klassifizierung kann anhand der folgenden beiden Dimensionen vorgenommen werden:

- Art des Produktmodells, die in Abhängigkeit der später zu bestimmenden Maße entsteht: So verlangen Größenmaße ein Größenmodell, Kopplungsmaße ein Kopplungsmodell etc.).
- Abstraktionsniveau, von dem aus das System betrachtet wird. Die verschiedenen Niveaus sind dabei durch die aufgrund der Enthaltenseinsrelation entstehende Hierarchie gegeben (z.B. Verzeichnis, Datei, Zeile).

Wie bereits in Kapitel 4.1.1 angedeutet, können diese Produktmodelle jeweils als Graphen dargestellt werden. Für den Zweck einer detaillierteren Darstellung werden hierfür *attributierte Graphen* verwendet (vgl. z.B. [Lewe88], S. 126f):

> Ein *attributierter Graph* ist ein 6-Tupel
>
> $\mathcal{G}_{att}$ = (*Knoten, Knotenfunktion, Knotenattribute, Kanten, Kantenfunktion, Kantenattribute*)
> mit
> einer *Knotenfunktion: Knoten* → *Knotenattribute*,
> einer *Kantenfunktion: Kante* → *Kantenattribute*,
> und den beiden endlichen Mengen *Knotenattribute* und *Kantenattribute.*

Definition 26: Attributierter Graph

Im folgenden soll das grundsätzliche Vorgehen des Abbildens der Produktmodelle auf attributierte Graphen vorgestellt werden und diese anhand der daraus resultierenden Anforderungen weiter konkretisiert werden:

- Die innerhalb der Produktabstraktion existierenden Objekte, bzgl. derer die Abstraktion vom Ausgangsprodukt vorgenommen wird und die im Mittelpunkt der Betrachtung der Produktabstraktion stehen (z.B. Klassen, Dateien, Subsysteme etc.) werden als Knoten repräsentiert. Die Knoten stellen folglich die typischen, für die Softwareproduktvermessung relevanten Entitäten dar.
- Innerhalb des Produktmodells existierende Relationen zwischen den in der Produktabstraktion existierenden Objekten werden auf binäre, gerichtete Kanten abgebildet. Mehrwertige Relationen können durch mehrere binäre Relationen dargestellt werden: So kann z.B. eine JAVA-Klasse durchaus mehrere Interfaces gleichzeitig implementieren. Diese n-äre Relation „implementiert n Interfaces“ kann allerdings durch mehrere binäre Relationen „implementiert ein Interface“ nachgebildet werden.
  Eine besondere, in Produktmodellen häufig verwendete Relation ist die Relation „ist Teil von“ (s.o.) zur Abbildung hierarchischer Strukturen.

Die Produktmodellerstellung überführt nun mittels der Produktabstraktionsfunktion das zugrundeliegende Softwareprodukt in einen solchen attributierten Graphen. Während der Vermessung (vgl. Kapitel 4) wird die Attributierung des Graphen wie folgt durchgeführt (dafür wird im folgenden mit n die Anzahl aller Softwaremaße bezeichnet):

- Jede zu vermessende Entität, d.h. eine bestimmte Menge von Knoten des attributierten Graphen, wird mit einem durch die Anwendung des Maßes m_i errechneten Meßwert attributiert. Da für einen Knoten $\mathcal{K}$ durchaus mehrere Maße angewendet und damit auch mehrere Meßwerte berechnet werden können, sind die Knotenattribute folglich Elemente des $\mathbb{R}^n$, auf die die Knotenfunktion abbildet, d.h. Knotenfunktion($\mathcal{K}$)→ $\mathbb{R}^n$ [14]. Der für ein Maß m_i errechnete Meßwert x_i eines Knotens $\mathcal{K}$ ist dabei durch den i-ten Wert seines Knoten-Tupels repräsentiert.
- Die Interpretation der Vermessung geschieht anhand der attributierten Knoten. Sämtliche statistischen Berechnungen werden dabei auf den von einem Maß erzeugten Knotenattributen durchgeführt.

Neben diesem allgemeinen Vorgehen besitzt das Erstellen und Vermessen – in Form der Attributierung – der Graphen für jede Maßklasse jeweils eigene, typische Charakteristika, die im folgenden für jede Klasse separat betrachtet werden sollen. Hierbei wird jeweils vom minimalen Produktmodell ausgegangen, das für die jeweilige Maßklasse gerade noch genügend Daten für die entsprechende Vermessung besitzt. Aufbauend auf dieser Analyse können grundsätzliche Probleme bei der Softwareproduktvermessung konkretisiert werden und ein entsprechender, in dieser Arbeit gewählter Lösungsversuch motiviert werden.

5.1.1 Vermessen von attributierten Graphen mit Größenmaßen

Die durch die Klasse von Größenabstraktionen erzeugten Graphen sind dadurch gekennzeichnet, daß sie nur gerichtete Kanten der Art „ist Teil von" besitzen. Die Kanten selbst besitzen keine Attribute. Die Vermessung von Entitäten kann nur für solche Knoten erfolgen, die prinzipiell noch enthaltene Knoten besitzen können. Ein Beispiel eines solchen Graphen für objektorientierte Systeme ist in Abbildung 18 gegeben. Das Vermessen eines Knotens (durch seine Attributierung) geschieht durch eine spezifische Summierung der im Knoten direkt oder indirekt enthaltenen Elemente. Die Vielzahl von Größenmaßen unterscheidet sich in der Art der minimalen, innerhalb des Produktmodells aufgespannten Enthaltenseinshierarchie und der spezifischen Art der Summierung. Der jeweils ermittelte Wert wird als Attribut des betrachteten Knotens verwendet.
Ein möglicher attributierter Graph des Maßes WMC ist als Beispiel in Abbildung 18 dargestellt (hierbei wird innerhalb der Methoden das Maß LOC verwendet, vgl. Kapitel 3.2.1).

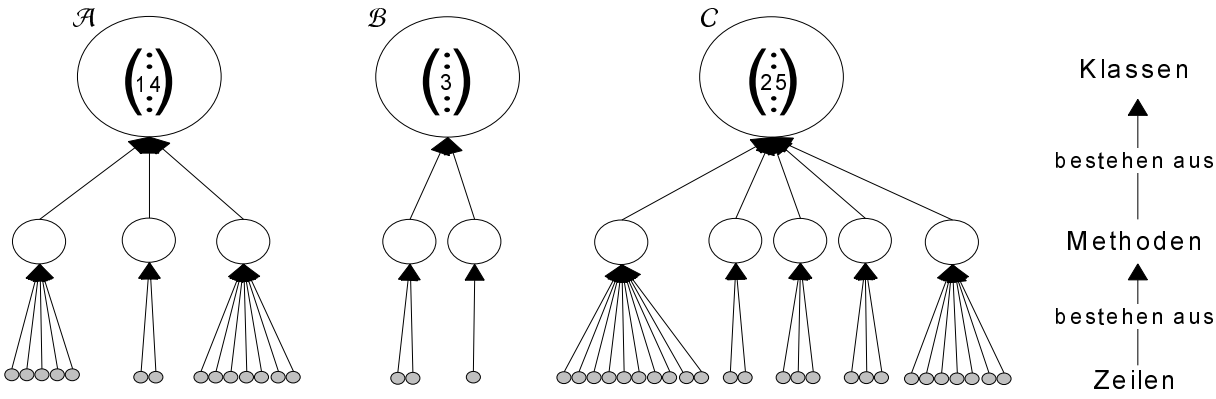


Abbildung 18: Attributierter Graph von Größenabstraktionen

[14] Genauer betrachtet sind die Knotenattribute Tupel von Elementen des jeweils verwendeten formalen Relationensystems. Da für diese im Bereich der Softwareproduktvermessung allerdings i.d.R. Elemente aus $\mathbb{R}$ verwendet werden (vgl. Kapitel 4.2), soll im folgenden $\mathbb{R}^n$ als Wertebereich der Knotenattribute angenommen werden.

Typische Charakteristika von WMC sind demnach:

- Eine Enthaltenseinshierarchie Klassen←Methoden←Zeilen,
- Vermessen als Aufsummieren der enthaltenen Elemente (Zeilen) der innerhalb der Klasse enthaltenen Elemente (Methoden).

Für das Beispiel ergeben sich damit für die Klassen $\mathcal{A}$, $\mathcal{B}$ und $\mathcal{C}$ die WMC-Werte 14, 3 und 25 (in Abbildung 18 angedeutet durch entsprechende Elemente der Attribute der die Klassen repräsentierenden Knoten).

5.1.2 Attributierte Graphen von Kopplungsabstraktionen

Die durch die Klasse von Kopplungsabstraktionen erzeugten Graphen sind dadurch gekennzeichnet, daß sie primär gerichtete Kanten der Art „ist gekoppelt“ besitzen (für eine genauere Beschreibung dieser Kanten vgl. Kapitel 7.2). Die Kanten selbst besitzen keine Attribute. Ein Beispiel eines solchen Graphen für objektorientierte Systeme ist in Abbildung 19 gegeben. Das Vermessen eines Knotens geschieht durch eine spezifische Summierung seiner eingehenden oder/und ausgehenden Kanten. Dabei ist es durchaus möglich, derartige Kanten ebenfalls von enthaltenen Elementen des betrachteten Knotens zu betrachten, wobei hierbei i.d.R. nur solche Kanten betrachtet werden, bei denen die betrachteten Elemente zu unterschiedlichen sie jeweils enthaltenen Knoten gehören. So werden z.B. für Klassen häufig Benutztbeziehungen betrachtet, die von ihren Methoden ausgehen, bei denen die beteiligten Methoden aber zu unterschiedlichen Klassen gehören (vgl. Kapitel 7.2). Die Vielzahl von Kopplungsmaßen unterscheidet sich in der Art der minimalen, innerhalb des Produktmodells aufgespannten Enthaltenseinshierarchie, der Art der betrachteten, die Kopplung repräsentierenden Kanten und in der spezifischen Art deren Summierung. Der jeweils ermittelte Wert wird als Attributierung des betrachteten Knotens verwendet. Ein möglicher attributierter Graph des Maßes CBO ist als Beispiel in Abbildung 19 dargestellt.

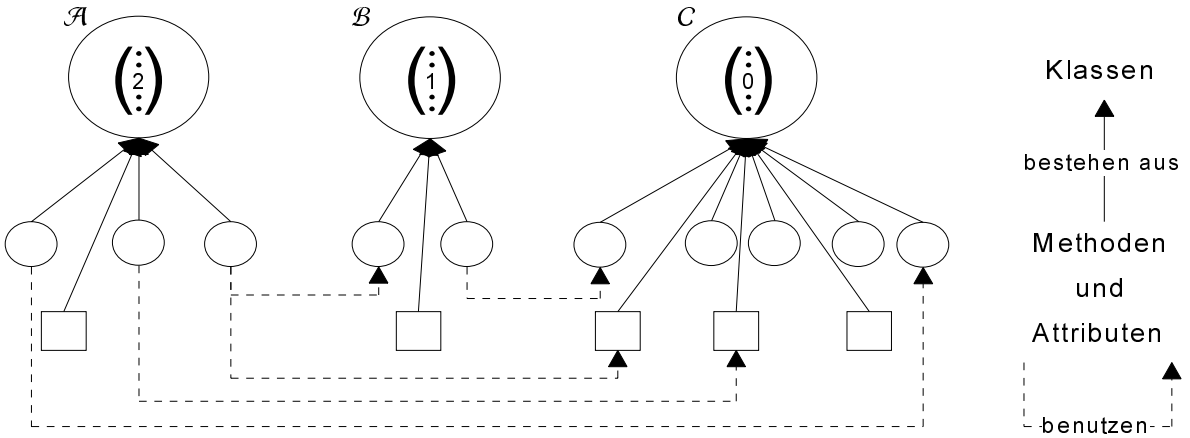


Abbildung 19: Attributierter Graph von Kopplungsabstraktionen

Typische Charakteristika von CBO sind demnach (vgl. Kapitel 3.2.2):

- Eine Enhaltenseinshierarchie Klassen←Methoden / Attribute,
- Betrachtung von Relationen der Art „benutzt“, d.h. eine Methode benutzt eine andere Methode oder ein Attribut, wobei die beiden Elemente der Relation zu unterschiedlichen Klassen gehören müssen, und
- Vermessen als Aufsummieren der unterschiedlichen, durch von Elementen der eigenen Klasse mittels Benutzt-Relationen direkt erreichbaren anderen Klassen. Direkt bedeutet hierbei, daß der Pfad der Benutzt-Relationen die Länge eins haben muß.

Für das Beispiel ergeben sich damit für die Klassen $\mathcal{A}$, $\mathcal{B}$ und $\mathcal{C}$ die Maße 2, 1 und 0 (wiederum angedeutet durch entsprechende Elemente innerhalb der Attribute).

5.1.3 Attributierte Graphen von Kohäsionsabstraktionen

Die durch die Klasse der Kohäsionsabstraktionen erzeugten Graphen stellen eine Kombination aus Größen- und Kopplungsabstraktionen dar: Entsprechend der Größenabstraktion wird jede zu vermessende Entität in Elemente verfeinert, zwischen denen dann jeweils entsprechend den Kopplungsabstraktionen zusätzliche Benutzungsrelationen eingefügt werden. Diese beschränken sich i.d.R. allerdings auf das übergeordnete, zu vermessende Objekt, d.h. es werden keine objektübergreifenden Objektrelationen eingefügt.

Ein Beispiel eines solchen Graphen für objektorientierte Systeme ist in Abbildung 20 gegeben. Das Vermessen eines Knotens geschieht durch eine spezifische Summierung seiner enthaltenen Elemente und der dazwischen existierenden Kanten. Die Vielzahl von Kohäsionsmaßen unterscheidet sich in der Art der minimalen, innerhalb des Produktmodells aufgespannten Enthaltenseins-Hierarchie, der Art der betrachteten, die Kopplung repräsentierenden Kanten und in der spezifischen Art der Summierung. Der jeweils ermittelte Wert wird als Attributierung des betrachteten Knotens verwendet.

Ein möglicher attributierter Graph des Maßes LCOM ist als Beispiel in Abbildung 20 dargestellt.

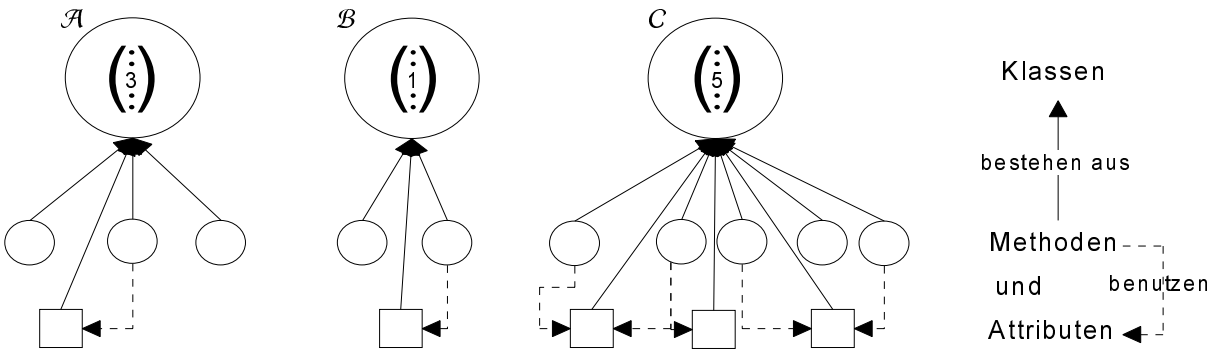


Abbildung 20: Attributierter Graph von Kohäsionsabstraktionen

Typische Charakteristika von LCOM sind demnach (vgl. Kapitel 3.2.3):

- Eine Enthaltenseinshierarchie Klassen←Methoden / Attribute,
- Betrachtung von Relationen der Art „benutzt Attribut“, d.h. eine Methode benutzt ein Attribut, wobei die Methode und das Attribut in derselben Klasse definiert sein müssen, und
- Vermessen als die Differenzbildung zwischen dem Aufsummieren der eigenen Methodenpaare, die kein Attribut gemeinsam benutzen und dem Aufsummieren der eigenen Methodenpaare, die wenigstens ein Attribut gemeinsam benutzen (vgl. Kapitel 7.1.2). Das Minimum von LCOM wird zusätzlich auf 0 festgesetzt.

Für das Beispiel ergeben sich damit für die Klassen $\mathcal{A}$, $\mathcal{B}$ und C die Werte 3, 1 und 5 (wiederum angedeutet durch entsprechende Elemente innerhalb der Attribute).

5.2 Anwendung des Graphenmodells

Das einheitliche Graphenmodell, das alle bisherigen Softwaremaße zu modellieren hilft, zeigt eine weitere Möglichkeit auf, die Meßwerte möglichst effektiv einzusetzen. Bei der Herleitung eines angepaßten Qualitätsmodells (vgl. Kapitel 2) wird eine relevante und zu bestimmende Qualitätseigenschaft in der Regel nicht in ein einzelnes Merkmal verfeinert werden, sondern in mehrere Merkmale, die sich zueinander unterschiedlich verhalten können (konkurrierend, verstärkend oder indifferent, vgl. Kapitel 2.1). Bei der Betrachtung der Meßwerte eines Maßes, das ein Merkmal charakterisiert, ist es daher für eine Interpretation häufig hilfreich, ebenfalls das Verhalten der anderen Meßwerte zu betrachten. Diese Möglichkeit der gleichzeitigen Betrachtung mehrerer Meßwerte einer Entität ist sehr einfach möglich durch die Vereinigung der einzelnen, den jeweiligen Maßen zugrundeliegenden attributierten Graphen.

Die Vereinigung mehrerer attributierter Graphen kann danach wie folgt durchgeführt werden:
Gegeben sind die einzelnen attributierten Graphen $\mathcal{G}_1,...,\mathcal{G}_h$ mit den Elementen (Knoten$_j$ (Abk.:$\mathcal{K}_j$), Knotenfunktion$_j$ (Abk.: $\mathcal{K}f_j$), Knotenattribute$_j$ (Abk.: $\mathcal{K}a_j$), Kanten$_j$ (Abk.: $\mathcal{KA}_j$), Kantenfunktion$_j$ (Abk.: $\mathcal{KA}f_j$), Kantenattribute$_j$ (Abk.: $\mathcal{KA}a_j$)) für einen Graphen $\mathcal{G}_j$, mit $j \in [1..h]$.
Der Vereinigungsgraph $\mathcal{G}_{gesamt}$, der wiederum ein attributierter Graph ist (vgl. Abschnitt 5.1), besteht aus den folgenden Elementen (die Bezeichnung $k \in$ Knoten$_j$ bezeichnet einen beliebigen Knoten in der Menge Knoten$_j$ des attributierten Graphen $\mathcal{G}_j$; entsprechendes gilt für die Kanten):

$\mathcal{K}_{gesamt} := \cup \mathcal{K}_j \quad j=1..h;$ [15]
$\mathcal{KA}_{gesamt} := \cup \mathcal{KA}_j ; \quad j=1..h;$

$$\mathcal{K}f_{gesamt} := \{(k,x) \in (\mathcal{K} \times \mathbb{R}^n) \mid k \in \mathcal{K}_j, 1 \le j \le h, x = \bigoplus_{1 \le j \le h,\, k \in K_j} \mathcal{K}f(k)\}$$

wobei $\oplus$ die einzelnen Komponenten x_i, y_i der zwei Tupel $x, y \in \mathbb{R}^n$ wie folgt addiert:

$x_i \oplus y_i :=$ x_i , wenn $x_i \neq$ <undefiniert>;
y_i , wenn $y_i \neq$ <undefiniert>;
undefiniert sonst.

Dabei bedeutet ein undefiniertes x_i, daß das entsprechende Maß m_i nicht auf das den Knoten beinhaltende Produktmodell angewendet wurde. Im Fall, daß sowohl x_i als auch y_i definiert sind, wurde das Maß m_i auf beide Produktmodelle angewendet. Aus der Annahme der internen Validität des Maßes folgt aber, daß gilt: $x_i = y_i$.

Die Kantenattribute und Kantenfunktion werden später definiert (s. Abschnitt 5.4).
Anhand dieses Modells von bisherigen Maßen kann Messen unabhängig der verwendeten Maße als *Attributierung von Knoten* aufgefaßt werden. Je mehr Maße auf einen Knoten angewendet werden, desto weniger Elemente innerhalb des $\mathbb{R}^n$ sind undefiniert.
Mit diesem einheitlichen Modell des Vermessens können folgende Probleme, die bereits in Kapitel 3.6.2 angedeutet wurden, konkretisiert werden:

- Wert-Rückprojektions-Problem: Da die Mächtigkeit des Konzepts Vermessung erst in größeren Systemen zum Tragen kommt, ist die Menge der Knoten sehr groß. Das Wert-Rückprojektions-Problem kann hierbei auf drei Teilschritte verfeinert werden:
  - Selektion interessanter Werte aus der Menge errechneter Meßwerte,
  - Identifikation der entsprechenden Knoten bzw. Entitäten des Produktmodells,
  - Identifikation der entsprechenden Softwareobjekte, zu denen die betrachteten Entitäten die Abstraktionen darstellen.

  Bei der Anwendung klassischer Meßwerkzeuge, die als Ergebnis i.d.R. Meßwertlisten liefern, bedeuten diese Schritte z.B.
  - Selektieren interessanter Listeneinträge. Dies geschieht z.B. durch das Ordnen der Listen und der Konzentration auf die j kleinsten bzw. größten Meßwerte.
  - Für jeden einzelnen Wert muß der entsprechende Knoten identifiziert werden. Das kann z.B. für die Vermessung von Klassen die Identifikation des Klassennamens und des Paketnamens, in dem die Klasse definiert ist, bedeuten.
  - Die im vorigen Schritt ermittelten Daten, die noch der Abstraktion des Produktmodells entstammen, müssen nun auf den Quelltext selbst übertragen werden. Für das

[15] Hierbei wird davon ausgegangen, daß Knoten unterschiedlicher Produktmodelle bzw. Graphen, die dieselbe Entität repräsentieren (z.B. Klassen), in der Vereinigungsmenge der Knoten nur einmal vorkommen. Liegt z.B. allen drei Graphen der Abbildung 18-Abbildung 20 derselbe Quelltext zugrunde, so bestünde der Vereinigungsgraph wiederum aus den drei Klassen $\mathcal{A}$, $\mathcal{B}$ und $\mathcal{C}$, 10 Methoden etc.

eingeführte Beispiel der Klassenvermessung sind die dafür notwendigen Schritte das Auffinden des entsprechenden Paketnamens innerhalb des Verzeichnisbaums, in dem der Quelltext abgelegt ist, das Auffinden der entsprechenden Datei, in der die Klasse definiert ist, und das Lokalisieren der Klasse innerhalb der Datei.

Alle diese Schritte müssen für jeden Meßwert einzeln durchgeführt werden und beruhen häufig auf dem mühsamen Finden von Entitäten entsprechend verwendeter Namensgleichheiten (z.B. das Auffinden des Verzeichnisses entsprechend des Paketnamens). Dieses Procedere ist für große Meßwertmengen nicht mehr effizient durchführbar, so daß der Vorteil von Softwaremaßen, schnell einen Überblick über gewünschte Informationen des Systems zu erhalten, zunichte gemacht wird.

- Multiwert-Problem: Das beschriebene Wert-Rückprojektions-Problem wird noch dadurch verschärft, daß für einen Knoten nicht nur ein Meßwert, sondern beliebig viele Meßwerte existieren können (vgl. *Multimetriken* in [ErLe96]), d.h. mehrere Elemente des $\mathbb{R}^n$ sind durch konkrete Meßwerte definiert. Jede Sortierung der Liste entsprechend eines bestimmten Maßes liefert eine neue Ordnung, in der wiederum jeder Eintrag entsprechend der Beschreibung im Wert-Rückprojektions-Problem auf das vermessene System zurück übertragen werden muß. Eine Interpretation von Werte-Kombinationen als Summe der Einzelinterpretationen ist auf diese Weise sehr ineffizient, aber dennoch häufig notwendig: Ein Beispiel dafür ist der konkurrierende Charakter der beiden Eigenschaften Kopplung und Kohäsion [Myer76]: Die Forderung nach maximaler Kohäsion und minimaler Kopplung ist nur als Kompromiß möglich, da kohäsive Systemteile, die um so kohäsiver sind, je kleiner sie gewählt sind (vgl. Kapitel 7), erst durch Kopplung an das Gesamtsystem angeschlossen werden können, die um so höher ist, je mehr Systemteile verknüpft werden müssen. Auch hier können Anomalien dieser Vermutung wichtige zusätzliche Interpretationen ermöglichen.
  Mögliche Verbesserungen wie Visualisierungen mit Kiviat-Diagrammen sind schwer auf große Mengen betrachteter Entitäten anwendbar; jedes Kiviat-Diagramm selbst besitzt nach wie vor das Wert-Rückprojektions-Problem.
- Wert-Interpretations-Problem: Das Betrachten eines konkreten Meßwertes geht vom entsprechenden Knoten aus. Der Kontext des Knotens, der für die Interpretation des Meßwertes i.d.R. sehr wichtig ist, da dort z.B. die Ursachen für den Meßwert liegen, bleibt verborgen. So wirft der Versuch einer Interpretation des Meßwerts 42 für eine Klasse bzgl. des Maßes „Anzahl von anderen Klassen, von denen eine Methode verwendet wird“ u.a. folgende Fragen auf:
  - Welche 42 Klassen werden benutzt?
  - In welchen Paketen liegen die verwendeten Klassen? Liegen alle Klassen in unterschiedlichen Paketen?
  - Wie intensiv ist die Verwendung der jeweiligen Klassen (wird nur eine Methode oder mehrere Methoden verwendet)?
  - Wie groß ist dieser Wert im Verhältnis zu anderen Klassen des betrachteten Systems?

Damit ergeben sich aus der auf diesem Modell basierenden Problemanalyse direkt folgende Lösungsansätze:

- Automatisierung der Rückprojektion eines Meßwertes auf den der Vermessung zugrundeliegenden Quelltext und dort auf die dem Meßwert zugrunde liegende Entität (z.B. Klasse, Methode etc.).
- Neue, effiziente und skalierende Darstellungsformen zur Präsentation von – mittels mehrerer Maße errechneten – Multi-Meßwerten mehrerer Entitäten.
- Kombinierte Darstellung von Struktur und Meßwert für eine detailliertere Analysemöglichkeit der Meßwerte.

Im folgenden Abschnitt sollen erste Realisierungen dieser Ansätze diskutiert werden. Die dort identifizierten Probleme werden anschließend eine Erweiterung des oben aufgeführten Maßmodells motivieren und damit den thematischen Rahmen für die folgenden Kapitel aufzeigen.

5.3 Kombination von Struktur und mehreren Meßdaten

Das Graphenmodell verdeutlicht ein wesentliches Problem bisheriger Meßansätze: Die vermessene Entität, d.h. der Knoten innerhalb des attributierten Graphen, enthält einen oder mehrere Meßwerte. Die gesamte Betrachtung der Meßwerte und deren Visualisierung erfolgt immer *knotenorientiert*, d.h. jeder Knoten wird separat behandelt und z.B. durch Kiviatdiagramme, Balkendiagramme, Tabellen usw. dargestellt. Messen abstrahiert folglich sehr stark vom zugrundeliegenden Produktmodell, d.h. an die Stelle der unterschiedlichen Kanten tritt lediglich ein numerischer Wert. Diese Isolation der einzelnen Knoten während der Meßwertbetrachtung begründet das Problem der Rückprojektion auf das System (vgl. Wert-Rückprojektions-Problem, Kapitel 3.6.2): Der Anwender hat dabei das Problem, die isolierte Entität mental wieder in das Gesamtsystem einordnen zu müssen. Wird z.B. eine Methode innerhalb eines objektorientierten Systems als zu lang identifiziert, so muß der Anwender, um diese Aussage analysieren und interpretieren zu können, den näheren Kontext der Methode kennen, d.h. die Klasse, zu der die Methode gehört, eventuelle Ober- und Unterklassen, die die Methode vererben bzw. erben, Client-Klassen, die eventuell die zu große Methode aufrufen usw. Das Problem der Rückprojektion einer vermessenen Entität in das zugrundeliegende System ist also durch das Fehlen von Strukturdaten der vermessenen Entität begründet.
Die Lösung dieses grundsätzlichen Problems der Informationsarmut von Meßwerten liegt in der gleichzeitigen Darstellung von Strukturinformationen und Meßdaten. Die dargestellte Struktur muß dabei wenigstens die vermessenen Entitäten beinhalten, damit diese bei der Darstellung um die Meßwerte angereichert werden können, d.h. die Knotenattribute des durch die Abstraktionsfunktion erstellten und durch das Maß attributierten Graphen müssen auf die Strukturdaten abbildbar sein.
Die gleichzeitige Darstellung von Meßwerten eines spezifischen Graphenmodells und Strukturinformationen kann dabei auf zwei verschiedene Weisen erfolgen:

- Durch Einblenden der Meßwerte in *eine* globale Strukturvisualisierung des gesamten Systems (*Monosicht*), d.h. Meßwerte eines spezifischen Graphenmodells werden in eine, vom Graphenmodell eventuell unabhängige, Strukturvisualisierung eingeblendet, oder
- durch das Anbieten *mehrerer*, i.d.R. auf unterschiedlichen Abstraktionsniveaus des Systems aufbauenden, mit Meßwerten angereicherten Strukturvisualisierungen (*Multisichten*), die zueinander konsistent sind und bei gleichzeitiger Betrachtung mehrerer dieser Sichten auf das System zueinander konsistent gehalten werden: Das Verändern der einen Sicht (z.B. die Fokussierung auf eine neue vermessene Entität) führt dabei automatisch zur Nachführung in den anderen Sichten (z.B. durch Hervorhebung der neuen Entität innerhalb des Vererbungsbaums).
  Jede Sicht repräsentiert dabei eine Integration von Meßwerten eines spezifischen Graphenmodells in eine spezifische Strukturvisualisierung. Multisichten können daher als Menge von Monosichten auf potentiell unterschiedlichen Abstraktionsniveaus aufgefaßt werden. Die unterschiedlichen Abstraktionsniveaus unterstützen sowohl das Top-Down- als auch das Bottom-up-orientierte Vorgehen beim Analysieren des Systems.

Im folgenden sollen beide Formen der integrierten Darstellung von Meß- und Strukturdaten exemplarisch vorgestellt werden. Eine Anwendung einer derartigen kombinierten Darstellung ist in Kapitel 3.3.3 beschrieben.

5.3.1 Monosichten

Innerhalb dieser Klasse von Meßwertdarstellungen wird versucht, eine globale Strukturvisualisierung des gesamten Systems derartig zu erweitern, daß gleichzeitig die Meßwerte der betrachteten Entitäten analysiert werden können. Die verschiedenen Möglichkeiten, die sich aus der Kombination unterschiedlicher Strukturvisualisierungen und unterschiedlicher Arten der Meßdateneinblendung ergeben, sind am weitesten in den Arbeiten von Lanza untersucht ([Lanz99], [DeDuLa99]). Die gleichzeitige Darstellung von Strukturinformationen und Meßdaten wird dort als *hybrider Ansatz* bezeichnet. Das Grundprinzip der meßwertangereicherten Strukturvisualisierung wird dort wie folgt umschrieben: „*We enrich a simple graph with metric information of the object-oriented entities it represents. In a two-dimensional graph we render up to five metrics on a single node at the same time.*“ ([DeDuLa99], S. 1).

Die Meßwerte werden dabei innerhalb der dargestellten Graphen auf verschiedene Attribute der dargestellten Knoten abgebildet: Knotenhöhe und -breite, X- und Y-Position des Knotens und die Knotenfarbe. In den meisten Fällen ist die X- und Y-Position allerdings bereits durch eine zugrundeliegende Systemstruktur belegt und wird durch Graphlayoutalgorithmen bestimmt, so daß nur drei verschiedene Meßdatenarten dargestellt werden können. So ist z.B. in Abbildung 21 ein normaler Vererbungsbaum eines Smalltalk-Projekts mit 166 Klassen dargestellt, bei dem die Knotenattribute Höhe, Breite und Knotenfarbe wie folgt gesetzt wurden ([DeDuLa99], S. 3f):

Knotenattribut	bestimmt durch das Maß[16]
Knotenhöhe	Anzahl der Methoden, die innerhalb der dem Knoten zugrundeliegenden Klasse definiert sind. Je höher der dargestellte Knoten, desto größer der Meßwert.
Knotenbreite	Anzahl der Attribute, die innerhalb der dem Knoten zugrundeliegenden Klasse definiert sind. Je breiter der dargestellte Knoten, desto größer der Meßwert.
Knotenfarbe	WMC mit LOC als Komplexitätsmaß der einzelnen Methoden. Je dunkler der dargestellte Knoten, desto größer der Meßwert.

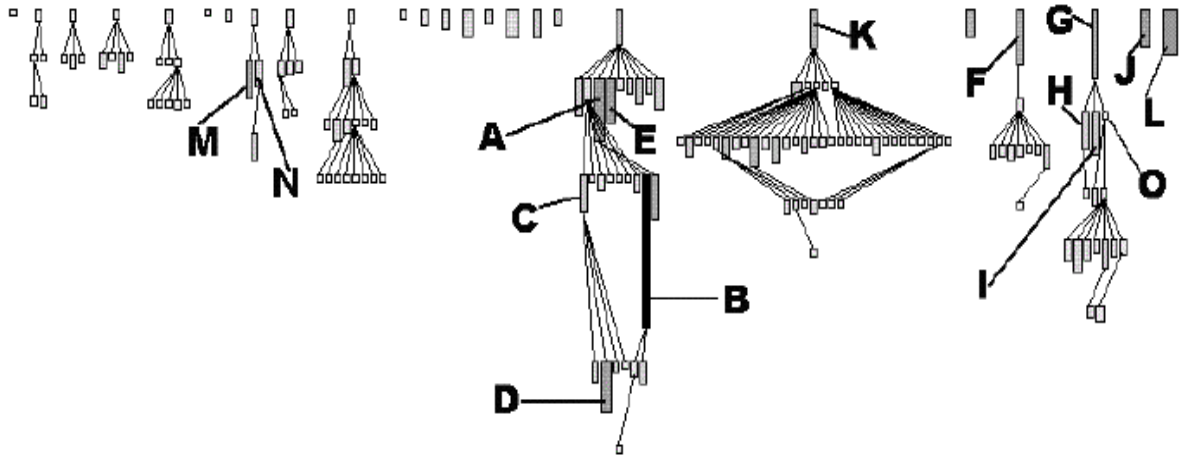


Abbildung 21: Mit Meßwerten angereicherter Vererbungsbaum, aus [DeDuLa99], S. 3

[16] Das in Kapitel 3.6.2 aufgezeigte Maß-Definitions-Problem gilt auch für diese Arbeit. Genauere Informationen zu den verwendeten Maßen und den ihnen zugrundeliegenden Produktmodellen können der Arbeit nicht entnommen werden. Inwieweit also z.B. geerbte Methoden oder Attribute berücksichtigt werden oder was als LOC aufgefaßt wird, muß an dieser Stelle unklar bleiben.

Innerhalb des vorgestellten, prototypischen Werkzeugs *CodeCrawler* können die Knotenattribute durch eine Auswahl aus 28 fest vorgegebenen Maßen beliebig bestimmt werden. Darüber hinaus existieren weitere Formen von Basisgraphen, in die Meßwerte eingeblendet werden können (vgl. [ebd.]), von denen allerdings nur noch dem *Konfrontationsgraphen* zur Darstellung von Benutztbeziehungen eine direkt durch das Produkt gegebene Struktur (z.B. durch Vererbung oder Benutztbeziehungen) zugrunde liegt. Die anderen Graphen stellen keine Systemstruktur mehr dar, sondern lediglich eine Vielzahl von Meßwerten (z.B. ein Histogramm- oder ein Korrelationsgraph).

Die mittels CodeCrawler erstellbare *Monosicht* zur gleichzeitigen Darstellung von Systemstruktur und Meßdaten bietet folgende Vorteile:

- Der Anwender erhält eine globale Sicht auf das System. Dabei ist die primär gewählte Struktur der Visualisierung dem Anwender i.d.R. bereits bekannt, da beispielsweise Vererbungsbäume oder Benutzungsgraphen klassische Hilfsmittel moderner Entwicklungsumgebungen sind. Die mentale Einbettung einzelner Elemente in das Gesamtsystem wird dadurch deutlich vereinfacht.
- Der Anwender ist in der Lage, ein angepaßtes Qualitätsmodell zu erstellen und die verfeinerten Merkmale mittels Maßen, so sie denn Teil der von CodeCrawler angebotenen Maßmenge sind, in die Struktursicht einzublenden.
- Die von CodeCrawler angebotene Darstellungstechnik ist sprachunabhängig; die der Berechnung der Maße zugrundeliegenden Produktmodelle lassen sich allerdings nur aus SMALLTALK-, C++- und JAVA-Programmen erzeugen.

Wesentliche Nachteile des Konzepts von Monosichten sind:

- Das Prinzip der Monosicht skaliert nicht beliebig. So zeigt die in Abbildung 21 dargestellte Menge von Entitäten (166 Klassen) bereits die Grenze der visuellen Auflösung auf. In [DeDuLa99] werden daher neben der in Abbildung 21 wiedergegebenen Globalsicht nur noch Teil-Vererbungsbäume betrachtet.
- Ein weiteres Skalierungsproblem von Monosichten betrifft die Anzahl der Entitäten innerhalb des einen verwendeten Abstraktionsniveaus. CodeCrawler z.B. bietet eine klassenbasierte Struktursicht an. Bei größeren Systemen mit über 1000 Klassen benötigt der Anwender eine über der Ebene der Klassen liegende Schicht, um eine Übersicht über das System zu erhalten. Bei Systemen dieser Größe liegt die Problematik der Rückprojektion gemessener Werte weniger im Auffinden der konkreten Klasse, als in der Identifikation der Menge von Klassen, die mit der vermessenen Klasse in einem funktionalen Zusammenhang stehen. Diese Aufteilung von Klassen in Mengen von Klassen ist in objektorientierten Systemen i.d.R. durch die Subsystemstruktur gegeben.
- Aufgrund der hohen Informationsdichte ist es nicht möglich, die einzelnen Entitäten mit ihren Bezeichnungen darzustellen. Innerhalb von CodeCrawler werden zwar je nach Mauscursor-Position weitergehende Informationen dargestellt (z.B. eindeutige Knotenbezeichnung), sie sind allerdings aufgrund der damit verbundenen Überdeckung anderer Informationen nur temporär. Der Anwender muß sich beim Analysieren des Systems die einmal identifizierten Entitäten merken. Die in Abbildung 21 dargestellte Einblendung einer Legende (vgl. Buchstaben) ist nicht Teil des Werkzeugs.
- Das Interpretieren der einzelnen produzierten Monosichten, die sich nur in den jeweils verwendeten Maßen und nicht in der für die Visualisierung verwendeten Systemstruktur unterscheiden, muß für jede Graph-Maß-Kombination neu erlernt werden. Nur nach einer jeweiligen Einarbeitungszeit ist es mittels dieser Darstellungen effizient möglich, Anomalien von vermuteten Verhältnissen zwischen Merkmalen zu identifizieren.

Zusätzliche Nachteile der durch CodeCrawler erzeugten spezifischen Monosicht sind:

- Die Abbildung der Meßwerte auf verschiedene Knotenattribute ist sehr ungenau. So ist es je nach Skalierung nur noch möglich, Extremwerte zu erkennen. Dies gilt besonders für das Knotenattribut Farbe: „[...] *the color metric is only useful for the detection of extreme values* " ([ebd]. S. 8).
- Das Problem der Rückprojektion eines Meßwertes auf den der Vermessung zugrundeliegenden Quelltext besteht nach wie vor: Wird während der Analyse der von CodeCrawler angebotenen Monosicht eine Klasse als mögliche Anomalie identifiziert, so muß diese in einem separaten Werkzeug erneut aufwendig identifiziert werden (vgl. Abschnitt 5.2).
- Das Layout der dargestellten Graphen ist durch kontextlose Graphlayoutalgorithmen berechnet, die versuchen, bestimmte Vorgaben wie z.B. minimale Kantenüberschneidungen zu optimieren (durch Bestimmung geeigneter X- und Y-Positionen der einzelnen Knoten). Daraus folgt, daß
  - die Darstellung nicht zwangsläufig *robust* gegenüber Änderungen des Produktmodells ist, da dessen kleine Änderung große Änderungen in der Darstellung zur Folge haben kann. Einmal identifizierte Strukturen müssen dann in der neuen Darstellung neu wiedergefunden werden.
  - die Anordnung der einzelnen Knoten nicht interpretierbar ist. Eine eventuell auffallende Sammlung von Knoten oder ein besonders außerhalb positionierter Knoten sind nur das Ergebnis eines Layoutalgorithmus und haben daher nicht notwendigerweise ein Pendant in der Systemstruktur.

Neben der Einblendung von Meßdaten in Vererbungsbäume ist auch die Anreicherung klassischer Klassendiagramme mit Meßdaten möglich. So wird in einem im Rahmen dieser Arbeit betreuten Projekt ein Ansatz vorgestellt und implementiert, bei dem UML-Klassendiagramme (vgl. z.B. [FoSc98]) durch Meßwerte angereichert werden [Przy99]. Hierbei wird lediglich die Farbe der dargestellten Klassen durch ein Maß bestimmt, das, da die Meßwertberechnung durch das Softwareprodukt-Meßwerkzeug Crocodile erfolgt, allerdings frei definierbar ist und daher den konkreten Verfeinerungen innerhalb des spezifischen Qualitätsmodells anzupassen ist. Die wesentlichen Nachteile von Monosichten gelten allerdings dort ebenso wie viele der im Kontext von CodeCrawler identifizierten, spezifischen Schwächen.

5.3.2 Multisichten

Die Grundidee der Multisichten besteht aus dem Anbieten mehrerer Sichten auf das System auf unterschiedlichen Abstraktionsniveaus und der Möglichkeit, diese zueinander konsistenten Teilsichten gleichzeitig zu betrachten und zu steuern. Jede Sicht auf jedem Abstraktionsniveau präsentiert dabei ein spezifisches Produktmodell auf einem spezifischen Abstraktionsniveau des Systems zusammen mit für dieses Produktmodel sinnvollen Meßwerten.
Im folgenden soll eine solche mit dem im Rahmen dieser Arbeit erstellten Werkzeug namens *CrocoBrowse* erzeugte Multisicht vorgestellt werden, deren praktische Validität zudem in Experimenten gezeigt werden konnte ([SiRuKö99a], [SiRuKö99b]):
Die dem System CrocoBrowse zugrundeliegende Meta-Struktur, die die Struktur derjenigen einzelnen Teilsichten strukturell beschreibt, die anschließend mit Meßwerten angereichert werden, kann wie in Abbildung 22 dargestellt werden:

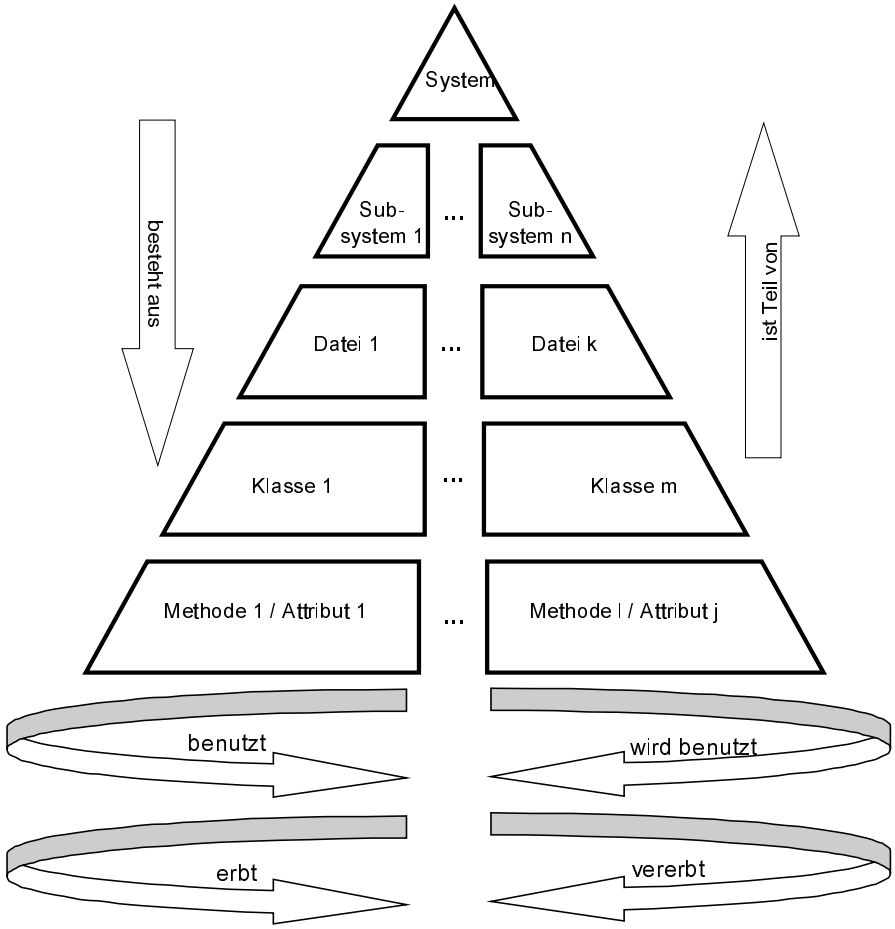


Abbildung 22: Metastruktur der Strukturdaten von CrocoBrowse

Für jede Entität eines jeweiligen Abstraktionsniveaus (System, Subsystem, Datei, Klasse, Methode/Attribut) werden die mit dieser Entität durch folgende Kanten verbundenen Knoten dargestellt:

- Alle Entitäten desselben Abstraktionsniveaus, die die aktuelle Entität benutzt. Dabei werden für die jeweiligen Abstraktionsniveaus spezifische Benutztbeziehungen definiert (eine Datei $\mathcal{A}$ benutzt z.B. eine andere Datei $\mathcal{B}$, wenn mindestens eine Methode der Klassen, die in Datei $\mathcal{A}$ definiert sind, mindestens eine Methode oder ein Attribut einer Klasse verwendet, die in Datei $\mathcal{B}$ definiert sind).
- Alle Entitäten desselben Abstraktionsniveaus, die die aktuelle Entität benutzen.
- Alle Entitäten desselben Abstraktionsniveaus, von denen die aktuelle Entität erbt. Dabei werden für die jeweiligen Abstraktionsniveaus Vererbungsbeziehungen definiert (eine Datei $\mathcal{A}$ erbt z.B. von einer anderen Datei $\mathcal{B}$, wenn mindestens eine Klasse, die in Datei $\mathcal{A}$ definiert ist, von mindestens einer Klasse erbt, die in Datei $\mathcal{B}$ definiert ist).
- Alle Entitäten desselben Abstraktionsniveaus, an die die aktuelle Entität vererbt wird.
- Die Entität des nächst höheren Abstraktionsniveaus, in dem die aktuelle Entität enthalten ist.
- Alle Entitäten des nächst niedrigeren Abstraktionsniveaus, die Teil der aktuellen Entität sind.

Für eine einfache Identifizierung wird jede Entität mit ihrem vollständigen Pfad entlang der Abstraktionsniveaus angegeben, so daß z.B. für jede Methode sofort klar wird, zu welcher Klasse sie gehört, in welcher Datei sie definiert ist und zu welchem Subsystem sie gehört.
Die durch diese Struktur aufgespannten Sichten auf das System sind in Form eines Hyperdokuments organisiert, d.h. jede entlang der Abstraktionsniveaus identifizierbare Entität

besitzt ein eigenes Dokument, in dem jeder aufgrund der oben aufgeführten Kanten angegebener Knoten und dessen Pfadelemente mit Referenz zum dazugehörigen Dokument angegeben sind. Übertragen auf das Graphenmodell ermöglicht CrocoBrowse folglich die Betrachtung von Knoten (und deren Attributen) und die Navigation entlang der vorgegebenen Kanten.
Jedes Abstraktionsniveau besitzt innerhalb von CrocoBrowse eine Konfigurationsdatei, die angibt, mit welchen Meßdaten die Entitäten auf dem entsprechenden Niveau angereichert werden sollen, d.h. wie die Knoten zu attributieren sind. Die Menge der verwendbaren Maße ist dabei unbegrenzt, da CrocoBrowse auf die Meßmaschinerie von Crocodile zugreift.
Um während der Analyse entlang des durch CrocoBrowse erzeugten Hyperdokuments zu keinem Zeitpunkt die Übersicht zu verlieren, werden mindestens die drei folgenden Sichten parallel und konsistent zueinander dargestellt (weitere Sichten vgl. Kapitel 9.1):

- *Projektbaumsicht*: Das Enthaltenseinsverhältnis der Entitäten innerhalb der verschiedenen Abstraktionsniveaus wird als Baum graphisch dargestellt. Die einzelnen Teiläste können dabei aus- und eingeblendet werden. Wird in einer anderen Sicht eine Entität selektiert, wird diese Selektion innerhalb des Baums nachgeführt, d.h. der relevante Ast bis zu der Ebene der selektierten Entität eingeblendet.
- *Referenztabellensicht:* Die Benutzt- und Vererbungsbeziehungen werden tabellarisch und jeweils mit Referenzen auf die entsprechenden Dokumente dargestellt. In diese Sicht werden darüber hinaus die Meßwerte eingeblendet: Je nach Konfiguration werden mehrere innerhalb des betrachteten Knotens angegebene Meßwerte dargestellt. Eine Identifizierung von Anomalien bzgl. der Merkmalsverhältnisse ist dadurch möglich. Ein Verfolgen von Entitäten entlang angebotener Beziehungen wird in der Baumdarstellung und der Quelltextsicht (s.u.) jeweils nachgeführt.
- *Quelltextsicht:* Auf Verlangen kann der einer Entität zugrundeliegende Quelltextteil ebenfalls automatisch dargestellt werden, d.h. die Rückprojektion ist äußerst effizient möglich. Der in der Quelltextsicht dargestellte Inhalt wird entsprechend möglicher Selektionen in anderen Sichten nachgeführt.

Die Vorteile, die die Monosicht bietet (vgl. Abschnitt 5.3.1), sind ebenfalls für die durch CrocoBrowse erzeugte Multisicht gültig:

- Der Anwender erhält eine globale Sicht auf das System. Dabei ist die primär gewählte Struktur der Visualisierung dem Anwender i.d.R. bereits bekannt, da beispielsweise Projektbäume ein klassisches Hilfsmittel moderner Entwicklungsumgebungen sind. Die mentale Einbettung einzelner Elemente in das Gesamtsystem wird dadurch vereinfacht.
- Der Anwender ist in der Lage, ein angepaßtes Qualitätsmodell zu erstellen und die verfeinerten Merkmale mittels Maßen, die ja durch Crocodile beliebig konfigurierbar sind, in die Struktursicht einzublenden.
- Da CrocoBrowse auf der Meßmaschinerie von Crocodile aufsetzt, ist CrocoBrowse für dieselben Sprachen anwendbar, für die Crocodile Produktmodellgeneratoren besitzt; dies ist für JAVA und C++-Quelltext der Fall. Das von CrocoBrowse erzeugte Hyperdokument ist mit einem Standard-HTML-Betrachter explorierbar und benötigt im Gegensatz zu CodeCrawler nach Fertigstellung der Sichten keine zusätzliche Software.

Darüber hinaus behebt das Konzept der Multisicht einen Großteil der Probleme, die bei der Monosicht beobachtet werden konnten (vgl. Abschnitt 5.3.1):

- Multisichten können sehr gut skalieren. CrocoBrowse-Strukturen z.B. sind erfolgreich für Projekte mit über 2000 Klassen angewendet worden. Aufgrund der verschiedenen angebotenen Abstraktionsebenen beginnt der Anwender mit einer Ebene, deren Elementanzahl noch übersichtlich scheint. Bei einer Projektgröße mit 2000 Klassen beginnt er

folglich i.d.R. nicht mit der Analyse auf der Ebene der Klassen oder Dateien, sondern auf der Ebene der Subsysteme.

- Aufgrund der gleichzeitigen Betrachtung mehrerer Sichten ist jede Entität zu jeder Zeit eindeutig identifizierbar. Wird z.B. innerhalb einer CrocoBrowse-Struktur eine Methode in der Referenztabellensicht näher untersucht, so ist in der Projektbaumsicht der weitere, aufgrund von Enthaltenseinsrelationen aufgespannte Kontext der Methode jederzeit ersichtlich.
- Die Menge der dargestellten Meßwerte beschränkt sich nicht auf das Abstraktionsniveau der in der Monosicht dargestellten Struktur. Jede Sicht und damit jede präsentierte Abstraktionsschicht kann eine eigene Menge von gemessenen Entitäten darstellen. Innerhalb von CrocoBrowse können folglich Subsystem-, Datei-, Klassen-, Methoden- und Attributmeßwerte dargestellt werden. Die dortige tabellarische Meßdatendarstellung erlaubt darüber hinaus ein sehr präzises Analysieren der gemessenen Werte.

Allerdings besitzen auch Multisichten einige Nachteile:

- Das gleichzeitige Betrachten mehrerer Sichten erfordert eine entsprechend große Darstellungsfläche auf dem Bildschirm. Werden die einzelnen Sichten zudem in separate Fenster abgebildet, treten die üblichen Probleme wie Überdeckung und Überlappung auf. Innerhalb von CrocoBrowse ist daher die Möglichkeit gegeben, die einzelnen Sichten durch separate Frames innerhalb einer HTML-Seite zu implementieren, für deren Darstellung, Aufteilung und Größe der Anwender größtenteils durch den darstellenden Browser unterstützt wird.
- Auch die Analyse von Multisichten muß für jede Kombination von Meßwerten und dargestellter Struktur neu erlernt werden. Selbst innerhalb von Crocodile, dessen grundlegende Struktur vorgegeben ist, vgl. Abbildung 22, benötigen unterschiedliche Konfigurationen der dargestellten Meßwerte unterschiedliche Analysemuster.

Spezielle Probleme der durch CrocoBrowse erzeugten Multisicht wie eine mangelnde Visualisierung der dargestellten Meßwerte oder die fehlende Darstellung von automatisch erkennbaren Funktionalitätsclustern sind im Folgeprojekt *CrocoCosmos* behoben (vgl. Kapitel 9.1).

5.4 Kantenattributierungen innerhalb des Graphenmodells

Die Informationsarmut des Graphenmodells zur Modellierung der Vermessung von Softwareprodukten begründet die Notwendigkeit, zusätzliche strukturelle Informationen in die Meßdaten einzublenden (s.o.). Die Aufgabe der Struktur ist, wenigstens diejenigen Knoten darzustellen, für die das Graphenmodell Attribute, also Meßwerte, anbietet. Die Art und Weise der Struktur zwischen den Knoten ist sehr einfach und durch eine geringe Anzahl von im Produktmodell abgelegten Kanten begrenzt; zudem ist die Art der Darstellung durch den für die Darstellung notwendigen Layoutalgorithmus bestimmt, d.h. das Layout der Darstellung darf nicht interpretiert werden. Damit werden wichtige Freiheitsgrade der Informationsdarstellung verspielt, da Meßwerte nur noch auf Darstellungsattribute des Knotens, nicht aber auf deren Position abgebildet werden können.
Speziell bei der Analyse großer Systeme spielt die Position der Knoten aber zwangsläufig eine sehr wichtige Rolle, da sie bei der Betrachtung eines Systemausschnitts (in Form eines Ausschnitts einer Mono- oder Multisicht) darüber entscheidet, welche Knoten überhaupt zusammen betrachtet werden können. Die durch Layoutalgorithmen zwangsläufig vorgenommene Partitionierung des Systems unterstützt daher nicht zwangsläufig das Verstehen eines Systems und verhindert teilweise sogar das Bilden eines mentalen Gesamtbildes der zu untersuchenden Software, das i.d.R. aus verschiedenen, bzgl. verschiedener Relationen in Verbindung stehender Teile besteht. Diese vom Menschen durchgeführte *Abstraktion durch Gruppierung* (vgl. Kapitel 6) ist ein wesentliches Merkmal bei der Bewältigung großer Datenmengen. Dabei werden häufig verschiedene Gruppierungen nebeneinander gehalten, da jede spezifische Stärken und Schwächen bei der Beantwortung von verschiedenen Fragen bzgl. der Daten besitzt. In Abbildung 23 sind z.B. drei häufig verwendete Gruppierungskategorien für den komplexen Datenbestand der Mail-Sammlung aufgeführt, die jeweils eine andere Sicht implementieren.

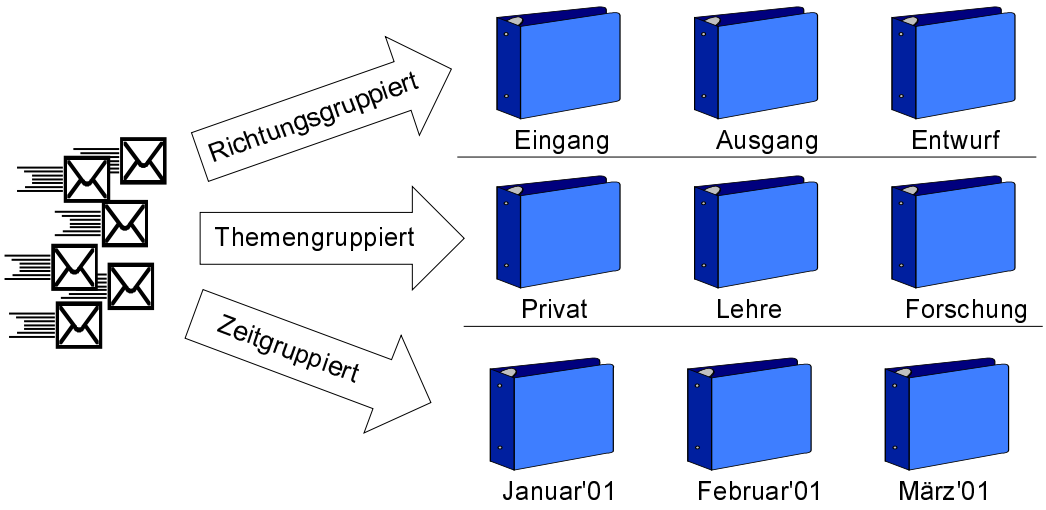


Abbildung 23: Beispielgruppierungen für die Verwaltung von Mails

Mittels derartiger Abstraktionen ist es möglich, beliebig große Datenmengen effizient zu verstehen. Dabei sind sowohl die Kriterien, nach denen gruppiert werden soll, als auch der Prozeß der Zuordnung eines Elements auf eine bestimmte Kategorie vom jeweiligen Ziel abhängig. In allen Fällen bieten sie aber eine hervorragende, anpaßbare Struktur, die mit Meßwerten angereichert werden kann. Solche Strukturen sind i.d.R. nicht direkt dem Produktmodell entnehmbar.

Um dieses Konzept generisch, d.h. für die verschiedensten Anforderungen an eine derartige Gruppierung geeignet, auch für Softwaresysteme zu unterstützen, wird in den folgenden Abschnitten dieser Arbeit eine Erweiterung des Graphenmodells behandelt: Zusätzlich zum Vermessen mittels bisheriger Maße, das als Attributierung der Knoten aufgefaßt werden kann, werden zwischen verschiedensten Elementen zusätzliche, bisher im Bereich der Softwareabstraktion und Softwarevermessung unbekannte Kanten hinzugefügt, deren Attribute anschließend für die Exploration verschiedenster Gruppierungen verwendet werden können. Die Kantenattribute werden verwendet, um die Zuordnung eines Elements zu einer Gruppe möglichst präzise bestimmen zu können. Damit ist es ebenfalls möglich, diese zusätzlichen Kanten als Verfeinerungen bisheriger, unattributierter Kanten zu verwenden (z.B. verschiedene, graduelle Ausprägungen der Relation „erbt von"). Die Attributierung der Kanten geschieht hierbei wiederum durch Vermessung. Erstmalig werden Meßdaten folglich nicht an Knoten, sondern an Kanten angefügt, d.h. ein Meßwert bezieht sich nicht mehr nur auf einen Knoten, sondern auf ein Paar von Knoten. Die für die Gruppierung einzufügenden Kanten sind dabei ungerichtet, da die Relation „gehört in dieselbe Gruppe wie" symmetrisch ist. Die Attributierung erlaubt darauf aufbauend ebenfalls eine detaillierte Analyse, zu welchem Grad ein Element zu welcher Gruppe gehört (d.h. es ergibt sich nicht zwangsläufig eine so strikte Trennung wie in Abbildung 23, die zudem, speziell in der themenbasierten Gruppierung, nicht eindeutig ist).
Der gewählte Ansatz beschränkt sich dabei nicht auf einzelne Kantenattributierungen, sondern beschreibt ein generisches Maß, dessen beliebige Instanziierungen jeweils zu neuen Kanten und Kantenattributierungen führen, d.h. innerhalb des attributierten Graphen sind auch die Kantenattribute durch Elemente des $\mathbb{R}^m$ gegeben; hierbei bezeichnet das i.-te Element des $\mathbb{R}^m$ den Meßwert des i-ten Kantenmaßes.
Die Anreicherung des Graphenmodells um solche Kanten und Kantenattribute erlaubt eine Visualisierung, bei der die Position der einzelnen Knoten untereinander entsprechend der gewählten Kantenattributierung interpretierbar ist. Damit ist es möglich, meßdatenbasierte Strukturen darzustellen, deren Wert über die Darstellung bloßer expliziter Strukturen (wie z.B. Vererbungsbaum oder Projektbaumhierarchie) hinausgeht und das Erstellen eines mentalen Bildes des Systems unterstützt. Diese meßdatenbasierten Strukturen erlauben erstmalig die intensive Beschäftigung mit *Kohäsion* (vgl. Kapitel 7), das als Grad des Zusammenhalts zwischen Elementen erst in der Kantenattributierung eine adäquate Entsprechung findet.
Das bisherige Vermessen als Knotenattributierung ist nach wie vor Bestandteil des Graphenmodells, d.h. die Kantenattributierung ist eine wirkliche Erweiterung und erlaubt die Integration aller bisherigen Maße (durch Beibehaltung der Knotenattribute).
Der Erweiterungscharakter äußert sich auch darin, daß es möglich ist, alle Graphdarstellungen, bei denen bisher Layoutalgorithmen die Positionen der Entitäten berechnen, durch eine meßbasierte Darstellung zu ersetzen. So bietet sich aufgrund einer Kantenattributierung die Möglichkeit, nicht nur die Knotenattribute in eine Strukturvisualisierung einzublenden, sondern ebenfalls die Kantenattributierung. Dies ermöglicht z.B. die meßdatenbasierte Positionierung von Klassen innerhalb von Klassendiagrammen oder die meßdatenbasierte Positionierung innerhalb von Vererbungsbäumen (vgl. Kapitel 8.3 und 8.4).

5.4.1 Bekannte Kantenattributierungen

In einigen wenigen Arbeiten ist die Idee der Kantenattributierung bereits angedacht worden. Hier sind besonders eine bereits in den neunziger Jahren vorgestellte Arbeit von Durnota und Mingins [DuMi92] und neuere Arbeiten von Geert Poels zu nennen [PoDe00]. In beiden Fällen werden die Kantenattributierungen aber lediglich dazu verwendet, neue Knotenattributierungen herzuleiten, d.h. die dort vorgestellte Vermessung entspricht zu einem gewissen Grad dem vorgestellten Graphenmodell bisheriger Maße: Die Vermessung selbst geschieht nach wie vor durch eine

Attributierung der Knoten. Lediglich für die Bestimmung des Meßwertes wird auf Summierungen von attributierten, je nach Ansatz unterschiedlichen Kanten zurückgegriffen.
So fokussiert sich die Arbeit von Durnota und Mingins auf das selten verwendete Qualitätsmerkmal *Kohärenz*: „*A class is coherent if the methods work together to carry out a single, identifiable purpose*“ ([DuMi92], S. 489). Dieses Knotenmerkmal wird durch attributierte Kanten zwischen den Methoden einer Klasse bestimmt. Die für diesen Zweck zwischen allen Methoden zusätzlich eingefügten Kanten besitzen eine Attributierung entsprechend der *Kohärenz-Distanz* zweier Methoden: „*The number of common methods called by two methods, divided by the total number of methods called by these two methods*“ ([ebd.], S. 492). Diese spezielle Attributierung, die in Kapitel 6 als Instanziierung eines allgemeinen *relativen Grads der Unähnlichkeit* bezeichnet wird, kann in ein spezielles Dendrogramm eingezeichnet werden, wie es allgemein z.B. für die Clusteranalyse verwendet wird (vgl. Kapitel 6.4.1). Aufbauend auf diesem Dendrogramm können verschiedene Kohärenzmaße für eine Klasse definiert werden. So ist z.B. die *intervallgewichtete Kohärenz* wie folgt definiert: „*The sum of weighted number of clusters in each ‚interval‘ of the dendrogram* [...]“ ([ebd.], S. 496). Die Gewichte werden dabei durch spezielle *Level* aus dem Intervall [0..1] gegeben, die die verschiedenen Iterationen eines agglomerativen Clusterverfahrens repräsentieren (vgl. Kapitel 6.4.1). Die Annahme, daß die Höhe des derart ermittelten Kohärenzwertes mit der Kohärenz korreliert, wird exemplarisch für zwei Klassen aus der Eiffel-Umgebung demonstriert (vgl. [ebd.]).
Eine andere Form der Kantenattributierung wird von Poels und Dedene verwendet [PoDe00]: Dort werden Softwareattribute allgemein als Distanz zu einem Referenzprodukt betrachtet: „*Every software attribute is defined as the distance from the software product that must be measured (or its abstraction) to some reference software product (or abstraction)*“ ([ebd.], S. 36). Diese Distanzen zu einem virtuellen Referenz-Produkt werden anschließend als klassische Knotenattributierung des betrachteten Softwareprodukts aufgefaßt. Der Vorteil des Umwegs über Distanzen ist, daß keine extensive Struktur und keine abgeschlossenen Operationen auf dem empirischen Relationensystem notwendig sind (vgl. Kapitel 4), da die durch Distanzen mögliche Intervallskala bereits genügend wertvolle Informationen liefert (vgl. [ebd.], S. 36f). Die Distanz selbst wird durch die Anzahl notwendiger, atomarer Schritte definiert, die für das vermessene Produkt (bzw. dessen Abstraktion) notwendig sind, um es in das Referenzprodukt zu überführen. Sowohl das Referenzprodukt als auch die atomaren Schritte sind dem jeweiligen Ziel der Vermessung anpaßbar.

Beide Arbeiten verwenden das in diesem Kapitel motivierte Konzept der Kantenattributierung, ohne wirklich neue Möglichkeiten der Softwareproduktvermessung vorzuschlagen und damit evtl. die aufgezeigten Problem zu bewältigen (vgl. Kapitel 3.6.2); statt dessen werden lediglich neue knotenbasierte Maße vorgestellt, die wiederum die oben aufgezeigten Probleme besitzen. Dennoch wird durch die beiden Arbeiten sowohl eine Technik demonstriert, deren Einsatz erst durch Distanzen möglich ist (Clusteranalyse), als auch grundsätzliche theoretische Vorteile der Distanzmaße aufgezeigt (Intervallskala). Diese Vorteile werden ebenfalls in dieser Arbeit ausgenutzt, wobei die Kantenattribute dann nicht lediglich zur Bestimmung von Attributen von Knoten höherer Abstraktionsstufen dienen.

Um Kantenattributierungen als Erweiterung bisherigen, knotenbasierten Messens grundlegend einzuführen, wird im folgenden Kapitel ein generisches Konzept der Kantenattributierung vorgestellt, das auf einem dem Menschen sehr vertrauten Distanz- bzw. Ähnlichkeitskonzept zwischen Entitäten aufbaut. In Kapitel 8 und 9 wird demonstriert, wie durch die Kantenattributierung als Erweiterung der knotenbasierten Vermessung die meßbasierte Qualitätssicherung großer Systeme ermöglicht wird.

5.5 Zusammenfassung

Die jedem Messen zugrundeliegenden Produktmodelle können durch attributierte Graphen modelliert werden. Das Vermessen innerhalb dieser Graphen mittels klassischer Softwareproduktmaße kann als Knotenattributierung aufgefaßt werden. Die Möglichkeit der Vereinigung mehrerer attributierter Graphen in einen Gesamtgraphen bildet die Voraussetzung zur Darstellung von Abhängigkeiten zwischen Merkmalen, die sich in Abhängigkeiten zwischen verschiedenen Meßwerten äußern.
Die grundsätzliche Problematik der Rückprojektion von meßwertbasierten Ergebnissen auf das zugrundeliegende System kann durch die zusätzliche Einblendung von systemimmanenter Struktur gelöst werden. Hierbei wird zwischen Monosichten, bei denen eine dem Anwender bekannte Globalsicht auf das System mit Meßwerten angereichert wird, und Multisichten unterschieden, bei denen Meßwerte mehrerer attributierter Graphen auf mehrere Strukturvisualisierungen auf unterschiedlichen Abstraktionsniveaus abgebildet werden und teilweise gleichzeitig dargestellt werden können.
Aufgrund der „Knotenlastigkeit“ bisheriger Maßansätze beschränkte sich diese Abbildung bisher auf das Abbilden von Knotenattributen. Die Anordnung der Knoten innerhalb der Struktur wird bisher durch Layoutalgorithmen vorgenommen. Zudem kann aus Gründen der Übersichtlichkeit hiermit nur eine Art von konkreten Kanten berücksichtigt werden. Erst eine Erweiterung der bisherigen Meßansätze um Kantenattributierungen gibt die Möglichkeit, durch das zusätzliche Abbilden dieser Attribute auf die Struktur meßdatenbasierte Positionierungen der dargestellten Knoten zu erzeugen. Die Interpretierbarkeit zusammen dargestellter Entitäten erlaubt damit das Bilden von Gruppierungen, um das System zu verstehen. Diese Art der Gruppierungen unterstützt das Konzept der Kohäsion. Der generische Charakter der Kantenattributierung erlaubt dabei unterschiedliche Gruppierungen und damit unterschiedliche Kohäsionsarten; die somit erstellbaren, unterschiedlichen Sichten auf ein System variieren bzgl. ihres Wertes je nach Ziel der Analyse des Systems.

„Denn, da sie [die Definitionen, Anm.d.A.]
Zergliederungen gegebener Begriffe sind,
so gehen diese Begriffe, obzwar nur noch verworren, voran,
und die unvollständige Exposition geht vor der vollständigen, so,
daß wir aus einigen Merkmalen,
die wir aus einer noch unvollendeten Zergliederung gezogen haben,
manches vorher schließen können,
ehe wir zur vollständigen Exposition,
d.h. der Definition gelangt sind."
(Kant in „*Kritik der reinen Vernunft: II Transzendentale Methodenlehre*")

6 Distanz- und Ähnlichkeitskonzept

Während in den vorigen Kapiteln das Vermessen als die Zuweisung von numerischen Werten an betrachtete Entitäten behandelt wurde, sollen in diesem Abschnitt sinnvolle, allgemeingültige Konzepte zur Bestimmung von Werten zwischen Entitäten, d.h. für die Kantenattributierung des im letzten Kapitel erarbeiteten Graphenmodells vorgestellt werden. Die Kantenattributierung sollte dabei mindestens folgende Anforderungen erfüllen:

- Sie sollte ähnlich wie die Knotenattributierung an die spezifischen Bedürfnisse des zugrundeliegenden Qualitätsmodells angepaßt werden können.
- Sie sollte unabhängig von der Knotenattributierung vorgenommen werden, um die bisherigen Meßkonzepte wirklich ausschließlich zu erweitern.
- Die anschließende Darstellung der berechneten Kanten sollte die Identifikation relevanter Informationen während der Anwendung von einem der in Kapitel 3.3 vorgestellten Prozesse erlauben. Dies schließt besonders die Skalierbarkeit auf praxisrelevante Systemgrößen ein (vgl. Relevanz-Problem in Kapitel 3.6.2).
- Die Kantenwerte sollten für eine einfache Interpretation eine intuitive Entsprechung besitzen. Während diese Anforderung z.B. für Größenmaße aufgrund des intuitiven Konzepts Größe häufig erfüllt ist, muß auf der Ebene der Kanten zwischen Entitäten erst nach einer Entsprechung gesucht werden (vgl. Wert-Interpretations-Problem in Kapitel 3.6.2).

Die bisherige Knotenattributierung und deren isolierte Darstellung trägt deswegen so wenig zur Bildung einer globalen Sicht auf das System bei, da die Stärke der menschlichen Wahrnehmung, sich bei geeigneter grafischer Aufarbeitung der Informationen mit wenig Aufwand auch innerhalb großer Objektmengen zurechtzufinden, durch die isolierte und für alle Entitäten homogene Darstellung von Meßwerten nicht entsprechend unterstützt wird. Die Einblendung der Meßwerte in eine Systemstruktur ist diesbezüglich als Verbesserung zu betrachten, allerdings weisen die bisherigen Systemstrukturdarstellungen einige Schwächen auf (vgl. Kapitel 5.3).
Um die Kantenattributierung in einer Weise durchzuführen, die es dem Menschen erlaubt, in effizienter Weise eine Übersicht über das System zu erhalten, ist es sinnvoll, Anleihen dafür der Psychologie zu entnehmen: Auf die Frage, wie Menschen bestimmte Wissensbereiche im Gedächtnis organisieren, werden sogenannte *semantische Gedächtnismodelle* erstellt, die „[...] *darzustellen suchen, wie sinnhaftes, d.h. semantisches Material gespeichert und abgerufen wird* " ([Maye79], S. 141). Hauptsächlich wird dabei zwischen folgenden beiden Modelltypen unterschieden ([Maye79], S. 141; [Holz01], Modul 2, Kapitel 4):

- *Netzwerk-Modelle*, die davon ausgehen, „[...] *daß Gedächtnisstoff aus Elementen besteht, welche untereinander in bestimmter Weise verknüpft sind* " (Maye79], S. 141) und

- *Merkmal-Modelle*, die auf der Annahme beruhen, „[...] *daß Gedächtnisinhalt sich zusammensetzt aus Merkmalen, welche Exemplare charakterisieren, und Merkmalen, welche übergeordnete Kategorien definieren* “ (Maye79], S. 141).

Das Netzwerk-Modell kann als eine Erweiterung des klassischen *Assoziationismus* betrachtet werden, der gewöhnlich auf die drei folgenden *aristotelischen Gesetze* zurückgeführt wird ([Maye79], S. 10):

1. *Gesetz der Kontiguität*, das besagt, daß „[...] *Ereignisse oder Objekte, welche gleichzeitig oder am gleichen Ort wahrgenommen werden, im Gedächtnis miteinander assoziiert* [sind] “,
2. *Gesetz der Ähnlichkeit*, das besagt, daß „[...] *Ereignisse oder Objekte, welche einander ähnlich sind, leicht miteinander assoziiert* [sind] “, und das
3. *Gesetz der Gegensätzlichkeit*, das besagt, daß [...] *Ereignisse oder Objekte, welche Gegensätze bilden, leicht miteinander assoziiert* [sind] “.

Im folgenden wird ein Ansatz vorgestellt, der beide Modelle miteinander verknüpft: Die innerhalb der Netzwerk-Modelle modellierten Assoziationen spiegeln dabei die zukünftigen Kanten zwischen den Knoten des Graphenmodells wider. Die Art der Assoziationen und deren Gewichtung soll dabei vor allen Dingen dem zweiten aristotelischen Gesetz der Ähnlichkeit genügen, d.h. Assoziationen zwischen zwei Entitäten sind um so stärker, je ähnlicher die Entitäten sind. Damit ist es möglich, das Gesetz der Kontiguität durch eine spezielle Instanziierung des Ähnlichkeitskonzepts zu subsumieren (z.B. ähnlich bzgl. des Wahrnehmungsortes oder der Wahrnehmungszeit). Das Gesetz der Gegensätzlichkeit kann als Inverses der Ähnlichkeit begriffen werden.
Innerhalb der Psychologie kann Ähnlichkeit zwischen Entitäten sehr gut mit der zweiten Klasse von semantischen Gedächtnismodellen erläutert werden: Die Merkmal-Modelle gehen davon aus, „[...] *daß jedes Konzept als eine Liste semantischer Merkmale im Gedächtnis gespeichert wird* “ (ebd. S. 149). Daher werden diese Modelle auch als *Attribut-Wert-Strukturen* bezeichnet ([Holz01], S. 68). Mittels eines Ähnlichkeitskonzepts innerhalb dieses Merkmal-Modells, das auf Überlappungen von Merkmalen der bzgl. der Ähnlichkeit zu untersuchenden Entitäten basiert, ist es möglich, „[...] *den Grad der Beziehung zwischen Wortpaaren einzuschätzen, um so ein Maß für „semantische Distanz“ zu gewinnen* “ ([Maye79], S. 149).
Im folgenden wird ein Ansatz vorgestellt, diese den Merkmal-Modellen entnommene semantische Distanz auf die Netzwerk-Modelle zu übertragen und dort entsprechend dieser Distanz gewichtete Assoziationen einzutragen. Für diesen Zweck wird im folgenden Abschnitt eine formalere Herangehensweise an das Konzept der Ähnlichkeit zwischen Entitäten vorgestellt.

6.1 Ähnlichkeit zwischen Entitäten

Für eine allgemeingültige Betrachtung von Ähnlichkeiten zwischen Entitäten werden im folgenden Arbeiten von Mario Bunge herangezogen, da die Zielsetzung seiner Onthologie, zugleich *exakt*, d.h. „[...] *built with the explicit help of logic or mathematics* “ ([Bung77], S. 8) und *wissenschaftlich*, d.h. eingebettet in das wissenschaftliche Geschehen heutiger Zeiten, zu sein, gute Voraussetzungen liefert, um allgemeingültige Konzepte zu finden, die jedoch konkret genug sind, um sie später in Software implementieren zu können. Diese informatiknahe Bearbeitung von Bunge war bereits häufiger Anlaß, Anleihen aus seiner Onthologie zu nehmen (z.B. [ChKe94], [AnCaLu98]).
Um eine merkmalbasierte Ähnlichkeit aufzuzeigen, muß noch einmal der bereits in Kapitel 4.1 aufgezeigte Unterschied zwischen Merkmal und Eigenschaft hervorgehoben werden: Während die *Merkmale* Charakteristika von Entitäten sind, die ungeachtet menschlicher Wahrnehmung und Wertung vorhanden sind („*A substantial property is a feature that some substantial individual possesses even if we are ignorant of this fact*“; [Bung77], S. 58), sind *Eigenschaften* Konzepte, deren Zuweisung

an die betrachtete Entität durch Menschen vorgenommen wird („[...] *an attribute is a feature we assign or attribute to some object: it is a concept*“, [Bung77]; S. 59). Die Verhältnisse zwischen Merkmalen und Eigenschaften können beliebig sein, d.h. die Merkmale können sich direkt auf eine, keine oder mehrere Eigenschaften beziehen. Dieser nicht-isomorphe Zusammenhang zwischen Eigenschaften und Merkmalen begründet das Scheitern der reinen Baum-Topologie beim Erstellen von Qualitätsmodellen (vgl. Kapitel 2.1).
Entsprechend Bunge besitzt jede Entität Merkmale. Die Menge aller Merkmale, die eine Entität besitzt, ist einmalig, d.h. es gibt keine zwei Entitäten mit exakt den gleichen Merkmalen (vgl. *principium individuationis* in Kapitel 4.1). Sowohl Bunge als auch die Arbeiten zu den Merkmal-Modellen innerhalb der Psychologie gehen nun davon aus, daß „[...] *the similarity between two things is the collection of their shared properties* “ ([Bung77], S. 85).
Damit ist es möglich, eine mengenbasierte Ähnlichkeitsfunktion zu definieren:

Sei $\mathbb{P}$ die Menge aller Merkmale.
Sei darüber hinaus die Funktion $p(x)$ für eine betrachtete Entität x definiert als

$p(x) = \{\mathcal{P}_i \in \mathbb{P} \mid x \text{ besitzt das Merkmal } \mathcal{P}_i\}$.

Dann ist die *Ähnlichkeit* σ zwischen den beiden Entitäten x und y durch den folgenden Ausdruck bestimmt:

$\sigma(x, y) := p(x) \cap p(y)$.

Definition 26: Mengenbasierte Ähnlichkeitsfunktion (nach [Bung77], Definition 2.15)

Da alle Entitäten bzgl. einiger Merkmale ähnlich sind (vgl. Theorem 2.3 in [Bung77]), kann es sinnvoll sein, die Menge der für die Ähnlichkeit betrachteten Merkmale zu beschränken. Werden darüber hinaus alle Merkmale mit der gleichen Gewichtung versehen, d.h. sind zwei Entitäten um so ähnlicher, je mehr gemeinsame Merkmale sie besitzen, so ist es damit möglich, eine Funktion zu erstellen, deren numerischer Funktionswert den Grad der relativen Ähnlichkeit zwischen zwei betrachteten Entitäten repräsentiert:

Sei $\mathbb{B}$ die Menge der für die Ähnlichkeit zu berücksichtigenden Merkmale, und sei $\mathbb{B}$ eine Teilmenge von $\mathbb{P}$, der Menge aller Merkmale.

Dann kann der *relative Grad der Ähnlichkeit bzgl. einer Merkmalsträgermenge* $\mathbb{B}$ zwischen zwei Entitäten x und y wie folgt definiert werden:

$$sim_{\mathbb{B}}(x,y) := \frac{|\sigma(x,y) \cap \mathbb{B}|}{|(p(x) \cup p(y)) \cap \mathbb{B}|} \quad .$$

Definition 27: Relative Ähnlichkeit bzgl. einer Merkmalsträgermenge $\mathbb{B}$ (nach [Bung77], Definition 2.16)

Aus dieser Definition lassen sich folgende Punkte ableiten, die bei dieser Betrachtung von Ähnlichkeit zwischen Entitäten wichtig sind, um einem möglichen Wert-Interpretations-Problem frühstmöglich zu begegnen (vgl. [SiLöLe99]):

- Ähnlichkeit bzgl. einer gegenüber den beiden Entitäten inkompatiblen Merkmalsträgermenge $\mathbb{B}$ ist nicht definiert. Eine Merkmalsträgermenge $\mathbb{B}$ ist demnach *ähnlichkeitsinkompatibel* zu zwei Entitäten, wenn keine ein Merkmal aus $\mathbb{B}$ besitzt.
- Ähnlichkeit hängt wesentlich von der Art der zugrundeliegenden Merkmalsträgermenge $\mathbb{B}$ ab. Für zwei betrachtete Entitäten kann deren Ähnlichkeit je nach gewähltem $\mathbb{B}$ alle möglichen Grade zwischen *identisch* und *gegensätzlich* annehmen.
- Ähnlichkeit ist keine Eigenschaft bzw. kein Merkmal an sich: Ohne eine explizite Beschreibung, welche Merkmalsträgermenge $\mathbb{B}$ der jeweiligen Ähnlichkeitsbestimmung zugrunde liegt, ist eine Ähnlichkeitsangabe unvollständig.

- Die der Ähnlichkeit zugrundeliegende Merkmalsträgermenge $\mathbb{B}$ enthält lediglich binäre Merkmale, d.h. entweder die Entität besitzt das Merkmal oder sie besitzt es nicht (vgl. Kapitel 4.1.2). Durch eine Diskretisierung kontinuierlicher Merkmale können diese allerdings in binäre Merkmale transformiert werden (vgl. dazu Abschnitt 6.3.2).
- Die relative Ähnlichkeit (vgl. Definition 27) entspricht eher dem intuitiven Ähnlichkeitsbegriff als die mengenbasierte Ähnlichkeit (vgl. Definition 26): So wird z.B. eine Entität x zu sich selbst als genauso maximal ähnlich angesehen wie eine Entität y zu sich selbst, auch wenn x sehr viel mehr Merkmale besitzt als y, d.h. $sim_{\mathbb{P}}(x, x)=sim_{\mathbb{P}}(y, y)$ obwohl $\sigma(x, x) > \sigma(y, y)$.
- Ähnlichkeit existiert lediglich zwischen mehreren Entitäten, d.h. die Ähnlichkeit einer einzelnen Entität ist nicht definiert. Für mehrere Entitäten kann der obige Ansatz auf beliebig viele Entitäten $x_1, x_2, ..., x_n$ ausgeweitet werden:

$$sim_{\mathbb{B}}(x_1, x_2, ..., x_n) := \frac{|(\bigcap_{i=1}^{n} p(x_i)) \cap \mathbb{B}|}{|(\bigcup_{i=1}^{n} p(x_i)) \cap \mathbb{B}|} .$$

  Da das hier vorgestellte Ähnlichkeitskonzept allerdings später für die Kantenattributierung zwischen zwei Knoten herangezogen werden soll, genügt für die weitere Arbeit das in Definition 27 vorgestellte Ähnlichkeitsmaß.
- Ähnlichkeit zwischen Entitäten ist symmetrisch, d.h. es gilt $sim_{\mathbb{B}}(x, y) = sim_{\mathbb{B}}(y, x)$. Diese Eigenschaft stimmt mit der intuitiven Vorstellung von Ähnlichkeit überein.
- Eine Entität ist zu sich selbst maximal ähnlich, d.h. $sim_{\mathbb{B}}(x, x)=1$. Auch diese Eigenschaft stimmt mit der intuitiven Vorstellung von Ähnlichkeit überein.
- Der Grad der Ähnlichkeit variiert zwischen den beiden Extrema *vollständig identisch bzgl. der Merkmalsträgermenge $\mathbb{B}$*, d.h. $sim_{\mathbb{B}}(x, y)=1$, und *vollständig unterschiedlich, also gegensätzlich bzgl. der Merkmalsträgermenge $\mathbb{B}$*, d.h. $sim_{\mathbb{B}}(x, y)=0$.

Die in Definition 27 aufgezeigte Ähnlichkeitsfunktion $sim_{\mathbb{B}}(x, y)$ wird ebenfalls innerhalb der multivariaten Statistik verwendet: Bei der Ähnlichkeitsbetrachtung von zwei Entitäten x und y bzgl. p binären Merkmalen wird eine sogenannte *Kontingenztabelle* erzeugt (s.u.), aus der ablesbar ist, wie häufig bei beiden Entitäten die Merkmalsausprägungen übereinstimmen bzw. wie häufig sie nicht übereinstimmen. Eine (positive) Übereinstimmung ist gegeben, wenn beide Entitäten das Merkmal besitzen (a_{xy}), und eine (negative) Übereinstimmung ist gegeben, wenn beide das Merkmal nicht besitzen (d_{xy}). Eine Kontingenztabelle für zwei Entitäten x und y bei p binären Merkmalen ist in der folgenden Tabelle dargestellt (vgl. [FaHa84], S. 377):

Merkmal ist	vorhanden bei Entität x	nicht vorhanden bei Entität x	Σ
vorhanden bei Entität y	a_{xy}	c_{xy}	$a_{xy} + c_{xy}$
nicht vorhanden bei Entität y	b_{xy}	d_{xy}	$b_{xy} + d_{xy}$
Σ	$a_{xy} + b_{xy}$	$c_{xy} + d_{xy}$	p

Der darauf aufbauende *S-Koeffizient* (*similarity coefficient*) ([FaHa84], S. 379), der definiert ist als

$$sc_{xy} := \frac{a_{xy}}{a_{xy} + b_{xy} + c_{xy}},$$

entspricht dabei in seinen Werten exakt der in Definition 27 aufgezeigten Ähnlichkeitsfunktion:

Es gilt: $a_{xy}+b_{xy}+c_{xy} = |p(x) \cup p(y)|$, da die linke Seite der Gleichung genau den Fällen entspricht, in denen wenigstens eine Entität das Merkmal besitzt; d_{xy} wird sowohl innerhalb des S-Koeffizienten als auch innerhalb des oben vorgestellten Ähnlichkeitskonzepts nie betrachtet.

Dieses Ähnlichkeitsmaß wird im folgenden als Grundlage für eine Kantenattributierung zwischen jeweils zwei Knoten herangezogen. Zuvor muß allerdings die Ähnlichkeit noch auf das Konzept der *Distanz* übertragen werden:

6.2 Distanzen zwischen Entitäten

Das im letzten Abschnitt eingeführte, merkmalbasierte Ähnlichkeitskonzept erlaubt, viele Phänomene aus der Psychologie zu erläutern (z.B. die Reaktionszeiten für Fragen der Art „*Rotkehlchen = Tier?* “, die sich umgekehrt proportional zur semantischen Distanz, d.h. proportional zur relativen Ähnlichkeit verhalten, vgl. [Maye79], S .149f).
In den folgenden beiden Abschnitten soll dieses Konzept dafür verwendet werden, ein Netzwerk-Modell der betrachteten Entitäten zu erstellen, d.h. die in Kapitel 5 angedachten Kantenattributierungen durchzuführen.
Für die weitere Verwendung des Konzepts Ähnlichkeit ist es sinnvoll, diese auf das Konzept der *Distanz* zu übertragen, da dieses, aufgrund des geometrischen Pendants, noch intuitiver ist, es eine direkte Visualisierung ermöglicht (vgl. Abschnitt 6.4) und eine Vielzahl von zusätzlichen statistischen Techniken ermöglicht (vgl. Abschnitt 6.4).
Grundsätzlich lassen sich Ähnlichkeiten sehr einfach in Distanzen transformieren, da i.d.R. nur die Forderung gestellt werden wird, „[...] *daß Objekte eine um so kleinere Distanz haben, je größer ihre Ähnlichkeit ist* “ [FaHa84], S. 376).
Häufig angewendet wird die folgende Transformation:

> Sei $sim(x, y)$ die Ähnlichkeit zwischen der Entität x und der Entität y und sei $0 \leq sim(x, y) \leq 1$,
>
> dann kann die *Distanz* $dist(x, y)$ definiert werden als
>
> $dist(x, y) := 1 - sim(x, y)$.

Definition 28: Distanz aus Ähnlichkeit (vgl. [FaHa84], S. 376)

Wird die Transformation auf das Ähnlichkeitsmaß aus Definition 27 angewendet, so entsteht ein *Distanzmaß,* das zudem metrisch ist (Beweis s.u.)*:*

> Sei $\mathbb{X}=\{x_1, x_2,...,x_n\}$ eine Menge von n Entitäten, von denen jede wiederum eine Menge von binären Merkmalen besitzt. Die Funktion d: $\mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}$ heißt *Distanzmaß*, wenn $\forall 1 \leq i \leq n,\ 1 \leq j \leq n$ gilt
>
> 1. $d(x_i, x_i) = 0$
> 2. $d(x_i, x_j) \geq 0$
> 3. $d(x_i, x_j) = d(x_j, x_i)$
>
> Wird zusätzlich die Dreiecksungleichung erfüllt, d.h. gilt $\forall 1 \leq k \leq n$
>
> 4. $d(x_i, x_k) \leq d(x_i, x_j) + d(x_j, x_k)$
>
> so heißt die Funktion d *metrisches Distanzmaß.*

Definition 29: (Metrisches) Distanzmaß (nach [FaHa84], S. 374)

Eine weitere, allgemeinere Möglichkeit, Ähnlichkeiten in Distanzen zu überführen, ist die Betrachtung des Inversen der Ähnlichkeit, der *Unähnlichkeit*. Zwei Entitäten besitzen demnach eine um so größere Distanz, desto unähnlicher sie sind.

Die Herleitung der entsprechenden Unähnlichkeitsfunktion entspricht daher dem Vorgehen aus Abschnitt 6.1:

> Die *Unähnlichkeit* zwischen den beiden Entitäten x und y ist durch die Menge der Merkmale bestimmt, die nur genau von einer der Entitäten besessen werden und kann wie folgt ausgedrückt werden:
>
> $\delta(x, y) := (p(x) \cup p(y)) \setminus^{17} \sigma(x, y)$.

Definition 30: Mengenbasierte Unähnlichkeitsfunktion (nach [Bung77], Definition 2.17)

Da alle unterschiedlichen Entitäten bzgl. einiger Merkmale unähnlich sind (vgl. [Bung77], S. 89), kann es sinnvoll sein, die Menge der für die Unähnlichkeit betrachteten Merkmale zu beschränken. Auch hier werden wieder alle Merkmale mit der gleichen Gewichtung versehen, d.h. zwei Entitäten sind um so unähnlicher, je mehr Merkmale nur genau von einer Entität besessen werden. Damit ist es nun möglich, eine Funktion zu erstellen, deren Funktionswert den Grad der Unähnlichkeit zwischen zwei betrachteten Entitäten repräsentiert.

> Sei $\mathbb{B}$ die Menge der Merkmale, die für die Unähnlichkeit berücksichtigt werden sollen und sei $\mathbb{B}$ eine Teilmenge von $\mathbb{P}$, der Menge aller Merkmale.
>
> Dann kann der *relative Grad der Unähnlichkeit bzgl. einer Merkmalsträgermenge* $\mathbb{B}$ zwischen zwei Entitäten x und y wie folgt definiert werden:
>
> $$dist_{\mathbb{B}}(x,y) := \frac{|\delta(x,y) \cap \mathbb{B}|}{|(p(x) \cup p(y)) \cap \mathbb{B}|} \ .$$

Definition 31: Relative Unähnlichkeit bzgl. einer Merkmalsmenge $\mathbb{B}$ (nach [Bung77], Definition 2.16)

Aus dieser Definition lassen sich alle Punkte ableiten, die schon bei der Betrachtung von Ähnlichkeit heraus gearbeitet wurden (vgl. Abschnitt 6.1).
Für dieses Maß soll im folgenden gezeigt werden, daß es sich um eine echte *Metrik* handelt, d.h. um ein metrisches Distanzmaß.

Beweis:

1) $dist_{\mathbb{B}}(x_i, x_i) = 0$ gilt, da $\delta(x_i, x_i) = \emptyset$, d.h. $|\delta(x_i, x_i)| = 0$
2) $dist_{\mathbb{B}}(x_i, x_j) \geq 0$ gilt, da Kardinalitäten von Mengen immer ≥ 0 sind.
3) $dist_{\mathbb{B}}(x_i, x_j) = dist_{\mathbb{B}}(x_j, x_i)$ gilt, da die Bildung der Vereinigung und der Schnittmenge kommutativ sind.
4) $dist_{\mathbb{B}}(x_i, x_k) \leq dist_{\mathbb{B}}(x_i, x_j) + dist_{\mathbb{B}}(x_j, x_k)$ kann erst nach einigen Vorarbeiten gezeigt werden:

Zu zeigen ist also:

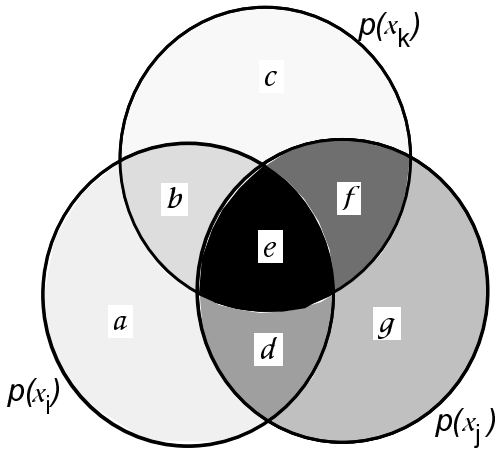
$$\frac{|\delta(x_i,x_k) \cap \mathbb{B}|}{|(p(x_i) \cup p(x_k)) \cap \mathbb{B}|} \leq \frac{|\delta(x_i,x_j) \cap \mathbb{B}|}{|(p(x_i) \cup p(x_j)) \cap \mathbb{B}|} + \frac{|\delta(x_j,x_k) \cap \mathbb{B}|}{|(p(x_j) \cup p(x_k)) \cap \mathbb{B}|}$$

Das Ausmultiplizieren mit den für alle ähnlichkeitskompatiblen Merkmalsträgermengen $\mathbb{B}$ positiven Nennern führt zu:

$$\begin{aligned} &|\delta(x_i, x_k) \cap \mathbb{B}| * |(p(x_i) \cup p(x_j)) \cap \mathbb{B}| * |(p(x_j) \cup p(x_k)) \cap \mathbb{B}| \\ &\leq \quad (|\delta(x_i, x_j) \cap \mathbb{B}| * |(p(x_i) \cup p(x_k)) \cap \mathbb{B}| * |(p(x_j) \cup p(x_k)) \cap \mathbb{B}| + \\ &\quad\ |\delta(x_j, x_k) \cap \mathbb{B}| * |(p(x_i) \cup p(x_k)) \cap \mathbb{B}| * |(p(x_i) \cup p(x_j)) \cap \mathbb{B}|) \end{aligned}$$

[17] Das „\" kennzeichnet hier die *boolsche Differenz*, die wie folgt definiert ist: $x \in (\mathbb{M} \setminus \mathbb{N}) \Leftrightarrow x \in \mathbb{M} \wedge x \notin \mathbb{N}$ ([Denv93], S. 1/6f)

Die einzelnen Faktoren können dabei entlang des folgenden *Venn-Diagramms* (vgl. [Denv93], S. 1/6), das für die drei Entitäten x_i , x_j , x_k sieben diskunkte Teilmerkmalsmengen (*a*, *b*, *c*, *d*, *e*, *f*, *g*) aufweist, wie folgt umformuliert werden (dabei bezeichnet a die Kardinalität von *a*, b die von *b* usw.):



- $|\delta(x_i , x_k) \cap \mathbb{B}|$ = a+d+c+f
- $|(p(x_i) \cup p(x_j)) \cap \mathbb{B}|$ = a+b+d+e+f+g
- $|(p(x_j) \cup p(x_k)) \cap \mathbb{B}|$ = b+c+d+e+f+g
- $|\delta(x_i , x_j) \cap \mathbb{B}|$ = a+b+f+g
- $|(p(x_i) \cup p(x_k)) \cap \mathbb{B}|$ = a+b+c+d+e+f
- $|\delta(x_j , x_k) \cap \mathbb{B}|$ = b+c+d+g

Abbildung 24: Venn-Diagramm für drei Mengen

Damit ergibt sich folgende zu beweisende Ungleichung:

(a+d+c+f)*(a+b+d+e+f+g)*(b+c+d+e+f+g)
≤ ((a+b+f+g)*(a+b+c+d+e+f)*(b+c+d+e+f+g)+
(b+c+d+g)*(a+b+c+d+e+f)*(a+b+d+e+f+g))

Das Ausmultiplizieren ergibt folgende Ungleichung:

$(a^2b+ab^2+a^2c+2abc+b^2c+ac^2+bc^2+a^2d+3abd+b^2d+3acd+3bcd+c^2d+$
$2ad^2+2bd^2+2cd^2+d^3+a^2e+2abe+2ace+2bce+c^2e+3ade+2bde+$
$3cde+2d^2e+ae^2+ce^2+de^2+a^2f+3abf+b^2f+3acf+3bcf+c^2f+4adf+$
$4bdf+4cdf+3d^2f+3aef+2bef+3cef+4def+e^2f+2af^2+2bf^2+$
$2cf^2+3df^2+2ef^2+f^3+a^2g+2abg+2acg+2bcg+c^2g+3adg+2bdg+$
$3cdg+2d^2g+2aeg+2ceg+2deg+3afg+2bfg+3cfg+4dfg+2efg+$
$2f^2g+ag^2+cg^2+dg^2+fg^2)$

≤ $(2a^2b+4ab^2+2b^3+2a^2c+6abc+4b^2c+2ac^2+2bc^2+2a^2d+7abd+5b^2d+5acd+6bcd+c^2d+$
$3ad^2+4bd^2+2cd^2+d^3+a^2e+5abe+4b^2e+4ace+5bce+c^2e+4ade+6bde+$
$3cde+2d^2e+ae^2+2be^2+ce^2+de^2+a^2f+6abf+5b^2f+5acf+7bcf+2c^2f+5adf+$
$8bdf+5cdf+3d^2f+3aef+6bef+4cef+4def+e^2f+2af^2+4bf^2+$
$3cf^2+3df^2+2ef^2+f^3+2a^2g+6abg+4b^2g+4acg+6bcg+2c^2g+5adg+7bdg+$
$5cdg+3d^2g+4aeg+6beg+4ceg+5deg+2e^2g+5afg+7bfg+5cfg+6dfg+5efg+$
$3f^2g+2ag^2+2bg^2+2cg^2+2dg^2+2eg^2+2fg^2)$

Durch komponentenweise Subtraktion der einzelnen Summanden auf jeder der beiden Seiten ergibt sich:

$$
\begin{aligned}
0 \quad \leq \quad & (a^2b+3ab^2+2b^3+a^2c+4abc+3b^2c+ac^2+bc^2+a^2d+4abd+4b^2d+2acd+3bcd+ \\
& ad^2+2bd^2+3abe+4b^2e+2ace+3bce+ade+4bde+ \\
& 2be^2+3abf+4b^2f+2acf+4bcf+c^2f+adf+ \\
& 4bdf+cdf+4bef+cef+2bf^2+ \\
& cf^2+a^2g+4abg+4b^2g+2acg+4bcg+c^2g+2adg+5bdg+ \\
& 2cdg+d^2g+2aeg+6beg+2ceg+3deg+2e^2g+2afg+5bfg+2cfg+2dfg+3efg+ \\
& f^2g+ag^2+2bg^2+cg^2+dg^2+2eg^2+fg^2)
\end{aligned}
$$

Da alle Summanden auf der rechten Seite ≥ 0 sind, folgt daraus die Behauptung. Q.e.d.

Da gilt: 1 - $sim_{\mathcal{B}}(x, y) = dist_{\mathcal{B}}(x, y)$ folgt daraus, daß auch 1 - $sim_{\mathcal{B}}(x, y)$ eine Metrik ist.

Metriken dieser Form sind in der Lage, die geforderte Kantenattributierung als echte Erweiterung der im Rahmen der Softwarevermessung bis jetzt vorgenommenen Knotenattributierungen durchzuführen. Aus dieser Erweiterung folgt für die künftige Nomenklatur allerdings, daß zwischen den beiden sonst synonym verwendeten Begriffen *Metrik* und *Maß* (vgl. z.B. die Titel von [FePf96], [Hend96], oder [OmPf97]) deutlich unterschieden werden muß. Während ein Maß Eigenschaften *einer Entität* in ein formales Relationensystem abbildet, bildet eine Metrik *Paare von Entitäten* auf ein formales Relationensystem ab. Im folgenden Abschnitt sollen mögliche Skalen unter Verwendung dieser Metrik vorgestellt werden.

6.3 Merkmalbasierte Skalen

Das vorgestellte, auf Merkmalen basierende Distanzmaß muß, damit die erhaltenen Metrikwerte analysierbar und interpretierbar sind und damit die Probleme der klassischen, knotenbasierten Vermessung frühstmöglich behoben werden können, wieder in einen Meßkontext eingebaut werden. Dafür gibt es zwei unterschiedliche Arten:

- Die der Metrikanwendung zugrundeliegenden Paare von Entitäten werden innerhalb des empirischen Relationensystems als zusammengehörende Entitäten aufgefaßt und der Metrikwert, der sich aus der Distanzbestimmung der beiden das Paar bildenden Entitäten bestimmen läßt, als Meßwert des Paares aufgefaßt. Statt n Meßwerte bei n Entitäten existieren dann n^2 Entitätspaare mit - aufgrund der Metrikeigenschaft - dementsprechenden (n(n-1)/2) Meßwerten. Aufgrund der Mächtigkeit möglicher intuitiver Aussagen über Ähnlichkeit kann dadurch eine *ordinalskalierte Entitätspaarvermessung* ermöglicht werden. Da entlang des Graphenmodells jeder Knoten eine Kante zu jedem anderen Knoten besitzt, wird diese Art der Vermessung im folgenden als *ordinalskalierte Kantenvermessung* bezeichnet.
- Das Distanzmaß kann ebenfalls zur Erstellung der für das intervallskalierte Messen geforderten quaternären Relation $(a_1 \times a_2)\ \Re\ (a_3 \times a_4)$ herangezogen werden (vgl. Kapitel 4.1.2), in dem das kartesische Produkt $(a_i \times a_j)$ innerhalb des empirischen Relationensystems die Bedeutung von Unähnlichkeit zwischen den beiden Entitäten besitzt und diese Eigenschaft mittels des vorgestellten Distanzmaßes abgebildet wird. Da für die Bildung einer Intervallskala zusätzlich eine einfache Ordnungsrelation zwischen den Entitäten Voraussetzung ist, müssen die für die Ähnlichkeit betrachteten Merkmale in Übereinstimmung mit den für die Ordnung zwischen den Entitäten herangezogenen Merkmalen sein. Damit ist es anschließend möglich, bisherige ordinalskalierte Knotenvermessung auf die Ebene der *intervallskalierten Knotenvermessung* anzuheben.

Beide Formen der Anwendung des Distanzmaßes für die Vermessung von Softwareprodukten werden im folgenden detailliert behandelt.

6.3.1 Ordinalskalierte Kantenvermessung

Die Beschäftigung mit Ähnlichkeit und deren numerische Bestimmung läßt sich in das klassische Feld der Meßtheorie (vgl. Kapitel 4) wie folgt einbetten:

Die Elemente des empirischen Relationensystems sind nicht mehr durch die einzelnen Entitäten, sondern durch die sie verbindenden Kanten gegeben. Die aufgezeigte Metrik läßt sich dadurch als normales Maß auffassen. Diese Möglichkeit der Kantenvermessung wurde bereits bei der Herleitung eines Graphenmodells für bisherige Meßansätze motiviert (vgl. Kapitel 5.4). Die anschließend zu identifizierenden Relationen sind folglich auch nur noch auf Kanten definiert.

Nach der Transformation einzelner Entitäten in Paare von Entitäten zur Betrachtung deren Ähnlichkeit läßt sich das Vorgehensmodell zur Vermessung von Softwareprodukten mittels intern valider Maße (vgl. Kapitel 4.4 und das dort aufgezeigte Vorgehensmodell zur Vermessung von Softwareprodukten mittels intern valider Maße) wie folgt anwenden:

1. Die problembasierte Festlegung des Maßeinsatzes sowie die Definition eines angepaßten Qualitätsmodells geschieht analog zu bisherigen Softwaremaßen: Wofür wird die Ähnlichkeitsbetrachtung angestellt und welche Ähnlichkeit wird konkret untersucht, um Aufschluß über welche darüberliegenden Qualitätsziele zu erhalten. Typische Instanziierungen sind in Kapitel 7 bis Kapitel 9 vorgestellt.
2. Die Verfeinerung der Eigenschaften bis auf die Ebene von Merkmalen geschieht auf der Ebene von *Merkmalsklassen*. Ziel dieses Schritts ist, die später für die Ähnlichkeitsuntersuchung verwendete Merkmalsträgermenge $\mathbb{B}$ zu identifizieren. Wichtig ist, daß, entsprechend der Erläuterungen in Abschnitt 6.1, alle späteren Aussagen nur bzgl. der verwendeten Merkmalsträgermenge gültig sind.

   Eine besondere Rolle kommt hier der Vermeidung von ähnlichkeitsinkompatiblen Merkmalsträgermengen zu: Die Vermessung von Kanten (d.h. ihre Attributierung, vgl. Kapitel 5) bedeutet, ihnen ein Wert zuzuweisen. Bei ähnlichkeitsinkompatiblen Merkmalsträgermengen ist dieser zumindest durch das vorgestellte Distanzmaß unbestimmt. Aufgabe dieses Schritts „Verfeinerung der Eigenschaften bis auf die Ebene von Merkmalen" sollte sein, solche Fälle weitestgehend auszuschließen, d.h. die Merkmalsträgermenge groß genug zu wählen. Da dies allerdings i.d.R. immer noch keine Garantie ist, müssen Paare von Entitäten für die gilt: $(p(x) \cup p(y)) \cap \mathbb{B} = \varnothing$ separat behandelt werden. Je nach verwendeter Merkmalsträgermenge kann eine der folgenden Setzungen sinnvoll sein:

   - Es kann sinnvoll sein, die Ähnlichkeit zweier Entitäten, bei denen keine wenigstens ein Merkmal aus der Merkmalsträgermenge besitzt, als vollständig unähnlich anzusehen. Innerhalb des formalen Relationensystems würde diese vollständige Unähnlichkeit auf die maximale Distanz, d.h. auf den Wert 1 abgebildet werden. Dieses Vorgehen ist besonders dann sinnvoll, wenn bei der späteren Betrachtung der primäre Fokus auf *Entitätsgruppen* liegt, d.h. auf Entitäten mit jeweils kleinen Distanzen zueinander (vgl. Kapitel 8), da solche als maximal unähnlich angesehenen Paare nicht fälschlicherweise zusammen betrachtet würden.
   - Es kann sinnvoll sein, Entitäten, die zu einer anderen Entität bzgl. der gewählten Merkmalsträgermenge nicht vergleichbar sind, grundsätzlich aus der Betrachtung auszuschließen. Dieses Vorgehen ist besonders dann sinnvoll, wenn es nur sehr wenige solcher Entitäten gibt. In den späteren Kapiteln wird sich zeigen, daß die Beschäftigung mit diesen „besonderen" Entitäten selbst wiederum wichtige Analysen ermöglicht, da diese Besonderheit als spezielle Anomalie betrachtet werden kann. So fallen z.B. leere Klassen, d.h. Klassen ohne Methoden, Attribute, Vererbung und Assoziationen aufgrund ihrer Merkmalsarmut (sie besitzen aber dennoch Merkmale, z.B. ihren Namen, ihr Erstellungsdatum usw.) bereits hierbei häufig auf. Diese Untersuchungsart kann auch explizit forciert werden durch die dritte Variante des Umgangs mit Entitäten, deren Ähnlichkeit bzgl. der gewählten Merkmalsträgermenge nicht bestimmbar ist:

- Für eine Herleitung einer *nominalskalierten Kantenmessung* kann es sinnvoll sein, Entitäten, die zu einer anderen Entität bzgl. der gewählten Merkmalsträgermenge nicht vergleichbar sind, als „*besonders*“ innerhalb des empirischen Relationensystems aufzufassen. Wird dieses „besonders“ auf einen „besonderen“, d.h. nicht durch andere Ähnlichkeitsbestimmungen verwendeten Wert wie z.B. (-1) abgebildet, ist anschließend eine nominalskalierte Vermessung möglich, die z.B. Aussagen erlaubt, wieviel Entitätspaare bzgl. der gewählten Merkmalsträgermenge vergleichbar sind und wieviele es nicht sind.

3. Die Definition der Abstraktionsfunktion, die Softwareprodukte in Softwareabstraktionen überführt, kann für den Fall der ordinalskalierten Kantenvermessung als Überführung jeder betrachteten Entität in eine Menge mit den zu betrachtenden Merkmalen angesehen werden.
4. Das Finden von Relationen bzgl. des Konzepts Ähnlichkeit ist aufgrund der intuitiven Vorstellung von Ähnlichkeit auf das Finden von ordnungbildenden, zweistelligen Relationen beschränkt: Da Aussagen bzgl. der Ähnlichkeit von Entitäten wie *„x ist zu y doppelt so ähnlich wie w zu z“* oder *„die Differenz der Ähnlichkeiten von x zu y und von w zu z ist kleiner als die Differenz der Ähnlichkeit von u zu v und von s zu t “* kaum Sinn machen, genügt hier das Erreichen der Ordinalskala. Es wird sich zeigen, daß bereits dieses Skalenniveau wertvolle Möglichkeiten der Softwarevermessung bietet (vgl. Kapitel 7, 8 und 9). Durch diese explizite Beschränkung auf ordinale Aussagen wird das Meßtheorie-Anwendungs-Problem deutlich reduziert, da die Anforderungen der Meßtheorie für diesen Skalentyp gering sind.
   Probleme bzgl. der Konsensfähigkeit der Relationen beruhen i.d.R. auf einer zu heterogenen Merkmalsträgermenge: Werden mehrere Klassen von Merkmalen in eine gemeinsame Merkmalsträgermenge übernommen, so werden Probleme mit bzgl. einer Merkmalsklasse nicht vergleichbaren Entitäten durch die eventuelle Vergleichbarkeit bzgl. einer anderen Merkmalsklasse, die ebenfalls Teil der Merkmalsträgermenge ist, überdeckt. Das folgende Beispiel soll das Problem verdeutlichen: Verschiedene Methoden einer Klasse können ähnlich sein bzgl. ihrer Attributsbenutzungen, ihrer Systemaufrufe, ihrer Assoziationen zu anderen Klassen oder auch bzgl. ihrer Namen oder ihrer Parameter. Während ein Konsens bzgl. der erstellten Ordnung basierend auf einer einzelnen dieser Merkmalsklassen sicherlich leicht möglich ist, ist dies für Kombinationen dieser Merkmalsklassen deutlich erschwert, da z.B. die vollständige Unähnlichkeit bzgl. einer Merkmalsklasse durch die Ähnlichkeit bzgl. einer anderen überdeckt wird. Ein Konsens bzgl. der Ordnung ist daher kaum möglich.
5. Das auf den vorigen Schritten aufbauende, empirische Relationensystem, das dann gegeben ist durch $< (\mathcal{A}\times\mathcal{A})$; *„ist bzgl. der Merkmalsträgermenge $\mathcal{B}$ mindestens so ähnlich wie“* $>$ läßt sich ohne weiteres mittels eines dazugehörigen formalen Relationensystems $< \mathbb{R} ; ' \leq ' >$ und des Maßes $dist_{\mathcal{B}}$ zu einer Ordinalskala ausbauen, die gegeben ist durch:
   $(< (\mathcal{A}\times\mathcal{A}) ;$ *„ist bzgl. der Merkmalsträgermenge $\mathcal{B}$ mindestens so ähnlich wie“*$>, < \mathbb{R} ; ' \leq ' >, dist_{\mathcal{B}})$.
6. Die Anwendung der Abstraktionsfunktion auf das zu betrachtende Produkt geschieht analog zum Vorgehensmodell der klassischen Softwaremaße.
7. Die Durchführung der Vermessung geschieht ebenfalls analog zum Vorgehensmodell der klassischen Softwaremaße. Hier zeigt sich, daß das gewählte Distanzmaß auf den recht einfachen Produktmodellen mit wenig Aufwand automatisiert angewendet werden kann, d.h. das Relevanz-Problem ist für diese Art der Vermessung kaum von Bedeutung.
8. Die Meßwertinterpretation kann natürlich normal entlang der im Kapitel 4.2.1 bis Kapitel 4.2.5. vorgenommenen Techniken geschehen. Ein wesentlicher Vorteil dieser Art von Meßwerten liegt allerdings in deren hervorragenden Visualisierbarkeit und der Anwendbarkeit zusätzlicher statistischer Verfahren. Spezielle Techniken dafür, namentlich die *dreidimensionale Visualisierung der Distanzen* und die *Clusteranalyse*, die ebenfalls besondere Vorgehensweisen einer Interpretation ermöglichen und damit maßgeblich das Multiwert-Problem und das Wert-Rückprojektions-Problem lösen helfen, werden im Abschnitt 6.4 vorgestellt.

9. Bei der Rückübertragung der Ergebnisse innerhalb des formalen Relationensystems auf das Produktmodell muß besonders berücksichtigt werden, daß sich die Ergebnisse immer auf Paare von Entitäten beziehen. Wie bei klassischen Maßen muß berücksichtigt werden, daß sich die Ergebnisse auf das Produktmodell beziehen; in diesem Fall bedeutet das die Einschränkung der Aussagen auf die zugrundeliegende Merkmalsträgermenge. Die Analyse, für welche Aspekte des dem Messen zugrundeliegenden Prozesses (vgl. Kapitel 3.3) die Resultate einer derart separierten Sicht in welchem Grad relevant sind, muß vom Anwender getroffen werden.

Wie bereits in den einzelnen Unterpunkten angedeutet, liegt der Schwerpunkt der weiteren Arbeit in genau dieser Form der Anwendung des erarbeiteten Distanzmaßes. Es wird ganz konkret gezeigt werden, wie die Kantenattributierung hilft, unterschiedliche Arten der Kohäsion objektorientierter Systeme zu bestimmen (Kapitel 8) und wie damit dem Kopplungs-Kohäsions-Problem begegnet werden kann, wie es möglich ist, aufbauend auf der Anreicherung bisheriger Meßtechniken mit diesen Distanzen wertvolle Visualisierungen anzubieten, mittels derer Systeme sowohl qualitativ bewertet werden können als auch bzgl. möglicher Restrukturierungen untersucht werden können. Die externe Validierung dieser Konzepte in Kapitel 9 belegt, daß damit die in Kapitel 3.6.2 aufgezeigten Probleme beim Produktmaßeinsatz deutlich reduziert werden können, so daß eine effiziente, meßbasierte Qualitätssicherung ermöglicht wird.

6.3.2 Intervallskalierte Knotenvermessung

Bei der Bestimmung der Merkmalsträgermenge für eine Betrachtung von Ähnlichkeit ist es ebenfalls möglich, die Knotenattributierungen, die durch klassische Softwaremaße vorgenommen werden, als Merkmale aufzufassen. Diese Idee ist besonders einfach an den Größenabstraktionen (vgl. Kapitel 5.1.1) zu erläutern:
Eine Größenabstraktion besteht aus den zu vermessenden Entitäten, die jeweils mittels einer „ist Teil von"-Hierarchie auf die bzgl. der Größe zu betrachtenden Elemente verfeinert sind. Größenmaße selbst summieren nur noch die enthaltenen Elemente. Dieses Größenmaß kann nun wie folgt in eine Merkmalsträgermenge überführt werden:

Innerhalb des durch die Größenabstraktion gegebenen Produktmodells sind die beiden Entitäten x und y auf die Mengen $\mathbb{X}$ und $\mathbb{Y}$ mit den Elementen $\{x_1, x_2, ..., x_n\}$ bzw. $\{y_1, y_2,...,y_m\}$ verfeinert, d.h. die x_i sind direkt oder indirekt Teil von x und die y_i sind direkt oder indirekt Teil von y. Für die Größenmaß-Bestimmung werden lediglich die Kardinalitäten von $\mathbb{X}$ und $\mathbb{Y}$ betrachtet. Gilt n=1 so kann die Menge gedeutet werden als *„x hat ein Element bzgl. dieser Größenbetrachtung"*. Gilt n=2 so kann die Menge gedeutet werden als *„x hat ein Element bzgl. dieser Größenbetrachtung"* und *„x hat zwei Elemente bzgl. dieser Größenbetrachtung"*. Für eine Menge mit n Elementen kann die Menge gedeutet werden als *„x hat ein Element bzgl. dieser Größenbetrachtung"*, *„x hat zwei Elemente bzgl. dieser Größenbetrachtung"* ... *„x hat n Elemente bzgl. dieser Größenbetrachtung"*. Wird diese Betrachtungsweise als Merkmalsmenge aufgefaßt und dieses für beide Mengen durchgeführt, d.h. $\mathbb{X}$ wird zu $\mathbb{X}_{Merkmal}$ und $\mathbb{Y}$ wird zu $\mathbb{Y}_{Merkmal}$, so entsteht auf Grundlage der Merkmalsträgermenge $\mathbb{B} := \mathbb{X}_{Merkmal} \cup \mathbb{Y}_{Merkmal}$ die Möglichkeit, ein auf den zugrundeliegenden Meßwerten basierendes Ähnlichkeitskonzept anzuwenden.

Inhaltlich ist diese Diskretisierung kontinuierlicher Merkmale in binäre Merkmale lediglich eine formalere Herangehensweise an die Differenz von Meßwerten. Sie kann für jede Instanziierung des Vorgehensmodells zur Vermessung von Softwareprodukten mittels intern valider Maße (vgl. Kapitel 4.4) zusätzlich verwendet werden, solange es sich um mindestens intervallskalierte Merkmale handelt und die für die Ähnlichkeit betrachteten Merkmale in Übereinstimmung mit den für die Ordnung zwischen den Entitäten herangezogenen Merkmalen sind. Ist z.B. das

Vorgehensmodell für das Größenmaß LOC vollständig instanziiert, so kann automatisch die Ähnlichkeitsbestimmung der betrachteten Entitäten bzgl. des LOC-Wertes vorgenommen werden. Inhaltlich ist diese Herangehensweise gleichzusetzen mit der Betrachtung von Verteilungsdiagrammen und Scatterplots, bei denen es ebenfalls um eine Übersicht über die vorkommenden Meßwerte geht, oder mit der Anwendung üblicher Statistiken oder Filterwerte (vgl. Kapitel 3.4).

Die an dieser Stelle vorgenommene, explizite Überführung dieses üblichen Vorgehens auf die merkmalbasierte Ähnlichkeit macht jedoch deutlich, daß auch klassische Softwaremaße für eine Kantenattributierung herangezogen werden können und damit auch ihnen die zusätzlichen Techniken der Visualisierung und Statistik zur Verfügung stehen. Hierzu zwei Beispiele:

- Es ist leicht möglich, Verteilungsdiagramme durch dreidimensionale Visualisierungen der Software zu ersetzen: Hohe Häufigkeiten bestimmter Meßwerte werden dann dort als Punktegruppierungen dargestellt, wohingegen Extremwerte auf isolierte Punkte abgebildet werden. Im Gegensatz zu den Verteilungsdiagrammen ist jedoch innerhalb der Ähnlichkeitsvisualisierung immer noch eine eindeutige Identifikation der einzelnen Entitäten möglich (vgl. Abschnitt 6.4.1), wodurch die Rückübertragung der Meßergebnisse auf die den Meßwerten zugrundeliegenden Entitäten vereinfacht wird (vgl. Wert-Rückprojektions-Problem).
- Es ist ebenfalls leicht möglich, statistische Filtertechniken wie z.B. das Aufzeigen der n größten/kleinsten Werte durch mächtigere Techniken wie die Clusteranalyse durchzuführen, in der Extremwerte erst sehr spät zu anderen Clustern hinzugefügt werden (vgl. Abschnitt 6.4.2). Die Möglichkeiten der Statistik, mittels verschiedener Perzentile Klassen von Meßwerten zu bilden, lassen sich ebenfalls sehr bequem mittels der Clusteranalyse durchführen und dort sogar entsprechend visualisieren.

Im folgenden wird diese Form der Anwendung der Metrik nicht weiter explizit behandelt, da sie sehr einfach verläuft; lediglich die damit zusätzlich möglichen Techniken der Visualisierung und Statistik werden in den folgenden beiden Abschnitten detailliert behandelt, da sie ebenfalls Grundlage der ordinalskalierten Kantenvermessung und deren weitere Anwendungen sind.

6.4 Techniken für Verwendung von Distanzmeßwerten

Das im letzten Abschnitt hergeleitete, auf Ähnlichkeit zwischen Entitäten basierende Distanzmaß kann für alle Entitäten einer zu betrachtenden Menge bestimmt werden. Für eine Menge mit n Entitäten ergeben sich daraus n^2 Distanzen, die häufig in einer sogenannten *Distanzmatrix* repräsentiert werden ([FaHa84], S. 374): Die einzelnen Elemente $d_{n,m}$ bezeichnen dabei die ermittelten Distanzwerte zwischen einer Entität e_n und einer Entität e_m. Diese Distanzmatrix hat folgende Eigenschaften:

- Sie ist symmetrisch, d.h. es gilt $d_{n,m} = d_{m,n}$. Dies folgt unmittelbar aus der Symmetrie des Distanzmaßes (vgl. Forderung 3 in Definition 29).
- Die Werte der Diagonale sind 0, d.h. $d_{i,i}=0 \ \forall \ i \in [1..n]$. Dies folgt unmittelbar aus der Forderung für Distanzmaße, zu sich selbst eine Distanz von 0 zu besitzen (vgl. Forderung 1 in Definition 29).

Die als Grundlage der Distanzmatrix ermittelten Ähnlichkeiten können analog zur Distanzmatrix in eine *Ähnlichkeitsmatrix* abgelegt werden ([FaHa84], S. 374). Auch sie ist symmetrisch, die Werte ihrer Diagonalen sind allerdings 1.

Das Arbeiten mit Ähnlichkeits- und Distanzmatrizen kann auf eine Vielzahl möglicher Techniken aufbauen, die als Standard in der multivariaten Statistik angesehen werden können. Für die Zwecke dieser Arbeit sind vor allen Dingen zwei Anwendungen interessant:

- *Clusteranalyse*: Hierbei geht es um das automatische Erkennen von Gruppen von Entitäten, die sich dadurch auszeichnen, daß sie untereinander jeweils eine geringe Distanz aufweisen, wohingegen die Elemente unterschiedlicher Gruppen jeweils eine große Distanz zueinander aufweisen. Diese Gruppierungen erlauben später das Betrachten einer großen Menge von Entitäten auf einem höheren Abstraktionsniveau, das durch die berechneten Gruppierungen gegeben sein kann.
- *Visualisierung*: Hierbei geht es um die Möglichkeiten der aussagekräftigen Visualisierung der betrachteten Entitäten und der zwischen ihnen errechneten Distanzen. Die Notwendigkeit der Visualisierung ist bereits in Kapitel 3.3.3 hergeleitet worden.

Im folgenden sollen beide auf Distanz- bzw. Ähnlichkeitsmatrizen basierende Techniken detailliert beschrieben werden.

6.4.1 Clusteranalyse

Die Ergebnisse des ähnlichkeitbasierten Distanzmaßes können direkt für die sogenannte *Clusteranalyse* verwendet werden[18]: "*The goal of cluster analysis is to divide a large data set into a number of sub-sets, called clusters, according to some given similarity measures* " ([Chen99], S. 61).
Die Möglichkeit, große heterogene Entitätsmengen in möglichst homogene Teilmengen aufzuteilen, erleichtert und verbessert die Analyse der ursprünglichen Daten: „[...] *a classification scheme may represent simply a convenient method for organizing a large set of data so that the retrieval of information may be made more efficiently* " ([Ever93], S. 2). Die aus der Clusteranalyse resultierenden Teilmengen sollen durch folgende Eigenschaften charakterisiert sein ([Ever93], S. 6f):

- *Interne Kohäsion*: Jede Teilmenge ist homogen bzgl. bestimmter Eigenschaften, d.h. die Entitäten einer Teilmenge sind einander hinsichtlich dieser Eigenschaften möglichst ähnlich.
- *Externe Isolation*: Die einzelnen Teilmengen sind paarweise verschieden bzgl bestimmter Eigenschaften, d.h. die Entitäten einer Teilmenge unterscheiden sich hinsichtlich dieser Eigenschaften deutlich von den Entitäten der anderen Teilmengen.

Allerdings ist diese Forderung nicht für alle Ausgangsdaten möglich, d.h. die durch die Clusteranalyse vorgeschlagenen Teilmengen bergen „[...] *a danger of interpreting clustering solutions in terms of the existence of distinct clusters even when in fact this is not the case* " ([Ever93], S. 6).
Da die Clusteranalyse, wie später auch die konkreten Visualisierungstechniken, Standardtechniken der multivariaten Statistik sind, sollen die verschiedenen existierenden Verfahren hier nur kurz vorgestellt werden. Für detailliertere Beschreibungen vgl. [Ever93], [FaHa84] oder [Chen99].
Während üblicherweise die Erläuterungen zur Clusteranalyse mit der Aufbereitung der Merkmalsausprägungen mehrerer betrachteter Entitäten beginnen, wird an dieser Stelle die Distanzmatrix, die durch die oben eingeführte Metrik bestimmbar ist, vorausgesetzt.
Ziel der Clusteranalyse ist nun, die Menge $\mathcal{G}$ der vermessenen Entitäten e_i , d.h. $\mathcal{G}= \{e_1\} \cup \{e_2\} \cup \{e_3\}... \cup \{e_n\}$, so in sich nicht überlappende $g \leq n$ Klassen C_k aufzuteilen, so daß gilt:

$$\left(\bigcup_{k=1}^{g} C_k = \mathcal{G} \right) \quad \wedge \quad \underset{k \neq j}{\forall} k, j \in [1..g]: C_k \cap C_j = \emptyset$$

Die einzelnen Klassen werden dabei als *Cluster*, eine konkrete Ausprägung der Aufteilung der Ursprungsmenge in verschiedene Cluster als *Partitionierung* bezeichnet.

[18] Andere Bezeichnungen für die Clusteranalyse sind: Klassifikationsverfahren, automatische Klassifikation, numerische Taxonomie, unsupervised learning sowie pattern recognition ([FaHa84], S. 371).

Die unterschiedlichen Techniken dieser *partitionierenden Clusterverfahren* lassen sich entsprechend ihres Vorgehens einteilen in (vgl. [FaHa84], [Löff99]):

- *Hierarchische Verfahren*, die eine Folge von Partitionierungen zwischen derjenigen Partitionierung, bei der sich alle Elemente in ein und demselben Cluster befinden, und derjenigen Partitionierung, bei der sich in jedem Cluster genau ein Element befindet, erzeugen. Während die *agglomerativen hierarchischen* Verfahren bei der letzteren Partitionierung anfangen und sukzessive Cluster zusammenfassen, gehen *divisive hierarchische* Verfahren genau umgekehrt vor.
- *Optimierungsverfahren,* die für eine vorgegebene Clusteranzahl diejenige Partitionierung ermitteln, für die eine gegebene Zielfunktion optimal ist. Diese Verfahren unterscheiden sich jeweils in der Art der Optimierung und der zu optimierenden Zielfunktion.
- *Stochastische Verfahren,* die die Zugehörigkeit der Entitäten zu einem Cluster als Zufallsvariable betrachten. Jeder Cluster ist demnach durch eine bestimmte, unbekannte Verteilung der Zufallsvariablen gekennzeichnet. Ausgehend von den gemessenen Merkmalsausprägungen der Entitäten werden die Parameter der Clusterverteilung geschätzt und die Zufallsfunktion ermittelt, anhand derer die Entitäten den Clustern, deren Anzahl im Vorfeld bekannt sein muß, zuzuordnen sind.

Am bekanntesten und für den Kontext dieser Arbeit am relevantesten sind die agglomerativen hierarchischen Verfahren (vgl. [Löff99]), da diese direkt auf der Distanzmatrix aufbauen und die verschiedenen Partitionen auch für große Datenbestände mit relativ geringem Rechenaufwand ermittelt werden können.
Ausgehend von einer Distanzmatrix $\mathfrak{D}$, den Entitäten e_1,..., e_n und der Menge $\mathcal{G}$ erzeugen agglomerative hierarchische Verfahren eine Folge von Partitionierungen $\mathcal{P}_n$, ..., $\mathcal{P}_1$. Die Partitionierung $\mathcal{P}_n$ besteht dabei aus n einelementigen Clustern, während $\mathcal{P}_1$ aus einem einzigen Cluster mit allen Entitäten besteht. Das Grundverfahren dieser Clusterverfahren kann nun wie folgt formuliert werden ([Ever93], S. 56f):

```
Start
     - Aufteilung aller Entitäten e_i in jeweils ein eigenes
       Cluster C_i
Solange Anzahl der Cluster größer 1 wiederhole:
     - Identifikation der beiden Cluster C_j und C_k mit
       minimalem Abstand zueinander.
     - Zusammenfassung der beiden Cluster C_j und C_k (z.B. durch
       Übernehmen der Elemente aus C_j nach C_k und
       anschließendem Löschen von C_j).
     - Dekrementierung der Anzahl der Cluster.
Ende
```

Das Vorgehen wird häufig durch ein sogenanntes *Dendrogramm* visualisiert, das in Abbildung 25 für eine Abarbeitung des beschriebenen Verfahrens für 5 Entitäten dargestellt ist. Die entstehenden Partitionierungen sind:

P_5= {{a}, {b}, {c}, {d}, {e}}
P_4= {{a, b}, {c}, {d}, {e}}
P_3= {{a, b}, {c}, {d, e}}
P_2= {{a, b}, {c, d, e}}
P_1= {{a, b, c, d, e}}

Die verschiedenen Arten der agglomerativen hierarchischen Clusterverfahren unterscheiden sich lediglich in der Art und Weise der Distanzberechnung zwischen Clustern für die Identifikation des Paares mit minimalem Abstand. Beispielhaft sollen hier kurz das *Single-Linkage-Verfahren*, das *Average-Linkage-Verfahren* und das *Verfahren von Ward* vorgestellt werden. Für weitere Techniken der Distanzbestimmung zwischen Clustern vgl. [Löff99] und [Ever93].

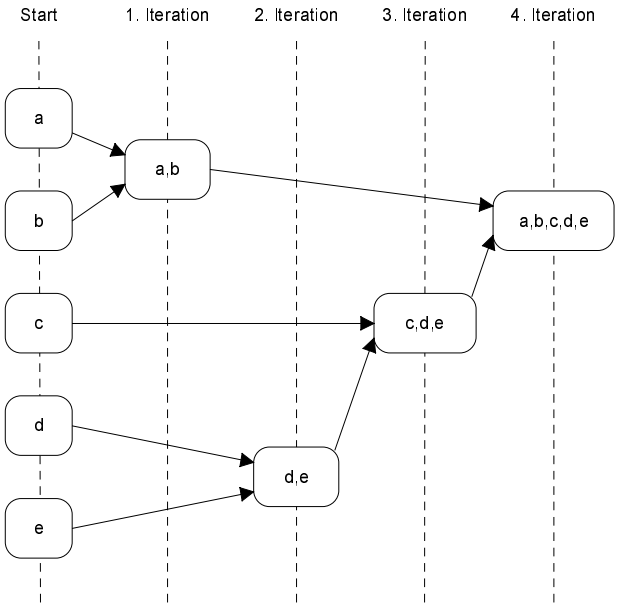


Abbildung 25: Dendrogramm (vgl. [Ever93], S. 56)

Beim *Single-Linkage-Verfahren*, das auch als „Nächster-Nachbar"-Technik bezeichnet wird, ist die Distanz zwischen zwei Clustern $\mathbf{C}_i$ und $\mathbf{C}_k$ definiert als die Distanz der beiden Entitäten e_l und e_m mit dem geringsten Abstand für die gilt: $e_l \in \mathbf{C}_i$ und $e_m \in \mathbf{C}_k$.

Innerhalb des *Average-Linkage-Verfahrens* ist die Distanz zwischen zwei Clustern $\mathbf{C}_i$ und $\mathbf{C}_k$ definiert als der Durchschnitt aller paarweisen Distanzen von Entitäten aus $\mathbf{C}_i$ und $\mathbf{C}_k$.

Beim Verfahren von *Ward* wird bei jedem Iterationsschritt versucht, den Informationsverlust, der durch das Zusammenführen zweier Cluster entsteht, zu minimieren. Dies wird dadurch erreicht, indem für alle möglichen Zusammenführungen der dadurch entstehende Informationsverlust berechnet wird und diejenige Zusammenführung mit dem geringsten Informationsverlust dann tatsächlich vorgenommen wird. Unter Informationsverlust versteht das Verfahren von Ward die Zunahme der Streuung innerhalb eines Clusters. Die Streuung kann dabei für einen Cluster $\mathbf{C}_j$ wie folgt berechnet werden:

$$Streuung(C_j) = \sum_{i=1}^{r} \| e_{iC_j} - \overline{e_{C_j}} \|^2$$

wobei e_{1C_j}, e_{2C_j}, ..., e_{rC_j} die Entitäten innerhalb des Clusters $\mathbf{C}_j$ sind und $\overline{e_{C_j}}$ den Durchschnitt der einzelnen Entitäten bzgl. ihrer Merkmalsausprägungen beschreibt.

Bereits hier wird deutlich, daß das Verfahren von Ward ursprünglich nicht auf der Distanzmatrix sondern auf der Merkmalsausprägung aufsetzt. In [HeHe95] wird dieses Verfahren allerdings derart modifiziert, daß es auf Grundlage der quadrierten Abstände berechenbar ist, da das „[...] *Varianzkriterium* [...] *auf natürliche Weise mit Abstandsquadraten zusammenhängt*" ([ebd.], S. 47). Beide Verfahren benötigen allerdings wenigstens intervallskalierte Meßwerte.

Auf Basis der Streuung wird die Homogenität einer Partitionierung als die Summe der Streuungen innerhalb der Cluster bestimmt. Da grundsätzlich gilt, daß die Homogenität einer Partitionierung $\mathcal{P}_n \leq \mathcal{P}_{n-1}$ ist, kann durch Subtraktion der Homogenität ($\mathcal{P}_n$) von der Homogenität ($\mathcal{P}_{n-1}$) ein Informationsverlust bestimmt werden. Beim Übergang von $\mathcal{P}_n$ zu $\mathcal{P}_{n-1}$ werden alle potentiellen $\mathcal{P}_{n-1}$ ermittelt, ihr Informationsverlust zur Partitionierung $\mathcal{P}_n$ bestimmt und diejenige mit dem geringsten Informationsverlust durchgeführt.

Bezüglich der verschiedenen Clusterverfahren kann keine allgemeingültige Wertung geschehen: „[...] *it should be clear that no one method can be judged to be ‚best' in all circumstance* " ([Ever93], S. 142). Für die praktische Anwendung der Clusteranalyse im Rahmen dieser Arbeit wurde daher ein Prozeß zur Selektion derjenigen Clustertechnik erarbeitet, die für den jeweiligen Kontext am

besten geeignet ist (vgl. [Löff99]). Die dafür notwendige, manuelle Nachkontrolle ist ein Standardschritt bei den agglomerativen Clusterverfahren, da auch für die Selektion einer geeigneten Clusteranzahl subjektive Kriterien verwendet werden. Der praktische Einsatz der Clusteranalyse zusammen mit diesen für die Durchführung notwendigen Schritten ist in Kapitel 9.3.1 beschrieben.
Eine weitere wichtige Technik für die Anwendung des Distanzmaßes, die zudem die Ergebnisse der Clusteranalyse analysieren helfen kann, ist im folgenden Abschnitt beschrieben

6.4.2 Visualisierung

Die im Kapitel 3.3.3 aufgestellten, allgemeingültigen Anforderungen an Visualisierung – bestehend aus der Forderung nach einem Alphabet und einer Grammatik – können auf Grundlage einer Ähnlichkeits- bzw. Distanzmatrix wie folgt erfüllt werden[19] :

- Das Alphabet ist durch die Menge der betrachteten Entitäten gegeben. Jede einzelne Entität soll anschließend als eigenständiges Objekt darstellbar sein.
- Die Grammatik ist durch die Elemente der Distanzmatrix gegeben. Jede Entität ist dadurch mit jeder anderen Entität verbunden, d.h. zwischen zwei Entitäten existiert jeweils eine (durch die Attributierung) gewichtete Kante.

Diese Art der Visualisierung großer Datenmengen mit dem Ziel, darin Strukturen zu erkennen, ist wichtiger Bestandteil des sogenannten *Information Retrieval*:

> „*Gegenstand des Information Retrieval ist die Repräsentation, Speicherung und Organisation von Information und der Zugriff auf diese Informationsressourcen.*“

Definition 32: Information Retrieval (aus [DäPa98], S. 3)

Eine häufige Möglichkeit zur Unterstützung des Information Retrieval ist die Erstellung *virtueller Informationsräume*, die im Gegensatz zum Darstellen physischer Datenräume auf abstrakten Raum- und Objektstrukturen basieren (vgl. [DäPa98], S. 40ff). Das allgemeine Vorgehen für deren Erstellung kann wie folgt beschrieben werden:

„*Grundsätzlich ergeben sich drei Schritte* [...] *zur Generierung virtueller Informationsräume:*

1. *Inhaltliche Erschließung des Datenraums und Bestimmung charakteristischer Objekteigenschaften,*
2. *quantitative Analyse der Objektähnlichkeiten auf der Grundlage ähnlicher Objekteigenschaften,*
3. *raumbezogene Darstellung der Objektrelationen (Mapping) nach der Assoziation: inhaltliche Nähe ≈ räumliche Nähe* “ ([ebd.], S. 55).

Damit kann die Visualisierung der vermessenen Kanten als spezielle Art der Erstellung eines virtuellen Informationsraums aufgefaßt werden. Die Schritte lassen sich dabei für den Kontext dieser Arbeit wie folgt konkretisieren:

1. Die betrachtete Software muß entsprechend der Ziele in ein Produktmodell überführt werden. Dabei müssen zumindest die Elemente der betrachteten Merkmalsträgermenge mit modelliert werden (Erschließung des Datenraums). Konkrete Instanziierungen dieses Schritts werden in Kapitel 7 und 8 vorgestellt.
2. Die quantitative Analyse beschränkt sich anschließend auf die Anwendung des vorgestellten Distanzkonzepts. Das Ergebnis dieses Schritts ist die Distanzmatrix.
3. Die Visualisierung geschieht durch das Abbilden der errechneten Distanzen auf geometrische Distanzen.

[19] Aufgrund der einfacheren Übertragbarkeit des Konzepts Distanz für eine Visualisierung (s.u.) wird die Aufgabe im folgenden auf die Visualisierung von Elementen mit dazugehöriger Distanzmatrix beschränkt.

Die Aufgabe 3 bedient sich dabei häufig des allgemeinen Problems des Darstellens ungerichteter Graphen, spezieller des Darstellens von *vollständigen Graphen*, d.h. Graphen, bei denen jeder Knoten durch eine gewichtete, ungerichtete Kante mit jedem anderen Knoten verbunden ist. Innerhalb der virtuellen Informationsräume werden die Kanten nicht mehr explizit eingezeichnet, da dies für größere Knotenmengen sehr unübersichtlich wird: So sind für n Knoten (n*(n-1))/2) Kanten darzustellen. Statt dessen werden die Kanten durch das Abbilden ihrer Gewichte auf räumliche Nähe dargestellt. Die Schwierigkeit dieses Verfahrens liegt darin begründet, daß für n darzustellende Objekte mit jeweiligen Distanzen dazwischen (n-1) Dimensionen notwendig sein können. Benötigt werden also Verfahren der *Dimensionsreduktion*:
„*However, if the information can be simplified to representations in one, two or three dimensions at the most, the human eye is usually capable of observing any differences and interactions between rows and columns with the aid of geometrical distance comparisons. This simplifying process is commonly known as multidimensional scaling* " ([ToStSt86], S. 105).
Grundidee ist, die wesentlichen Daten des n-dimensionalen *Distanzmodells* in ein euklidisches *Raummodell* niedrigerer Dimension zu überführen ([FaHa84], S. 673). Die wesentlichen Daten innerhalb des Distanzmodells sind die Ordnungen der vermessenen Entitäten (vgl. Abschnitt 6.3.1), d.h. die innerhalb des Distanzmodells gegebene Ordnung, daß zwei Entitäten x und y dichter beieinander liegen als zwei Entitäten v und w, soll auf die geometrische Distanz im Raummodell überführt werden. Der Wunsch, die nach einer Dimensionsreduktion errechneten Positionen später darstellen zu können, führt dazu, daß in der Praxis hauptsächlich zwei- oder dreidimensionale Raummodelle verwendet werden.
Im wesentlichen werden zwei praktisch relevante Arten unterschieden, wie diese Transformation vorgenommen wird (vgl. [DäPa98], Kapitel 3; [FaPoFu99], S. 193ff):

1. Berechnung der Positionierung von Objekten im zwei- oder dreidimensionalen Raum mit Hilfe von *nichtlinearen Optimierungsverfahren*, die häufig in der Physik verwendet werden, und
2. Durchführung der *Hauptkomponentenanalyse* zur Berechnung einer Projektion in den zwei- oder dreidimensionalen Raum unter größtmöglicher Beibehaltung der Varianz der Ausgangsdaten.

Beide Techniken sollen im folgenden kurz vorgestellt werden:

Nichtlineares Optimierungsverfahren
Als ein typisches, nichtlineares Optimierungsverfahren soll hier die *Spring-Embedder-Technik* vorgestellt werden: Diese Technik wurde 1984 erstmals von Eades vorgestellt [Eade84] und „[...] *is now one of the most popular algorithms for drawing undirected graphs with straightline edges, widely favored in information visualisation systems for its simplicity*" ([Chen99], S. 71.). Das grundsätzliche Modell von Eades verwendet eine Metapher aus der Physik: Jede vermessene Entität wird als Ring modelliert, der zufällig im zwei- oder dreidimensionalen Raum positioniert wird. Zwischen den Ringen werden Federn gespannt, deren Länge im idealen, energielosen Zustand exakt der jeweiligen Distanz entspricht, die aus der Distanzmatrix für die beiden den Ringen entsprechenden Entitäten abgelesen werden kann. In Abbildung 26 ist eine solche Startkonfiguration für drei Entitäten e_1, e_2 und e_3 mit den jeweiligen Federn f_1, f_2 und f_3 dargestellt (die Länge der Federn im entspannten Zustand ist jeweils durch die Anzahl der Federwicklungen angedeutet). Die Idee des Spring-Embedder-Verfahrens liegt nun darin, dieses Startsystem *ausschwingen* zu lassen, d.h. für alle an den Federn durch Dehnung oder Stauchung auftretenden Kräfte ein Energieminima zu bestimmen: Dabei sind für jede Feder drei zueinander disjunkte Zustände zu unterscheiden:

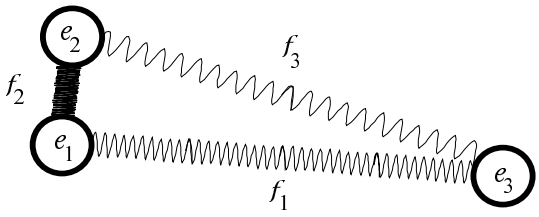


Abbildung 26: Startkonfiguration eines Spring-Embedders

1. Die Feder befindet sich im spannungslosen Zustand (z.B. die Feder f_1 in Abbildung 26), d.h. die Distanz zwischen den durch die Feder verbundenen Entitäten entspricht der Länge der Feder im spannungslosen Zustand. Da letztere der berechneten Distanz aus der Distanzmatrix entspricht, bedeutet dies, daß die dargestellte Distanz im euklidischen Raum der errechneten Distanz entspricht. Für das Verfahren folgt daraus, daß die Positionen der beiden Entitäten bzgl. dieser Feder nicht verändert werden müssen.
1. Die Feder befindet sich im gestauchten Zustand (z.B. die Feder f_2 in Abbildung 26), d.h. die Distanz zwischen den durch die Feder verbundenen Entitäten ist kleiner als die Länge der Feder im spannungslosen Zustand. Da letztere der berechneten Distanz aus der Distanzmatrix entspricht, bedeutet dies, daß die dargestellte Distanz im euklidischen Raum kleiner als die berechnete Distanz ist. Für das Verfahren folgt daraus, daß die Entfernung zwischen den beiden Entitäten bzgl. dieser Feder vergrößert werden muß.
2. Die Feder befindet sich im gedehnten Zustand (z.B. die Feder f_3 in Abbildung 26), d.h. die Distanz zwischen den durch die Feder verbundenen Entitäten ist größer als die Länge der Feder im spannungslosen Zustand. Da letztere der berechneten Distanz aus der Distanzmatrix entspricht, bedeutet dies, daß die dargestellte Distanz im euklidischen Raum größer ist als die berechnete Distanz. Für das Verfahren folgt daraus, daß die Entfernung zwischen den beiden Entitäten bzgl. dieser Feder verkleinert werden muß.

Das Ergebnis dieses Verfahrens angewendet auf die Startkonfiguration in Abbildung 26 ist in Abbildung 27 dargestellt: Alle Federn befinden sich im spannungslosen Zustand.

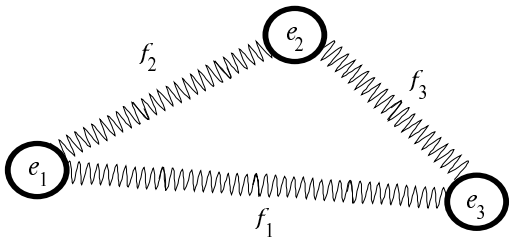


Abbildung 27: Ergebnis eines Spring-Embedders.

Allgemeiner kann das Spring-Embedder-Verfahren wie folgt formuliert werden: Wird eine Feder gestaucht oder gedehnt besitzt sie Energie. Diese Energie kann berechnet werden durch den Ausdruck $\mathcal{W}=½\mathcal{K}d^2$. Dabei stellt $\mathcal{K}$ die sogenannte *Federkonstante* dar (die groß ist für „starke" Federn und klein für „schwache" Federn und als Charakteristikum einer Feder gegeben ist) und d beschreibt die Differenz zwischen der Länge der Feder innerhalb des Systems und der natürlichen Länge der Feder im spannungslosen Zustand.

Für den dreidimensionalen Raum, in dem ein Punkt $\mathcal{P}_i$ die Koordinaten x_i, y_i, z_i besitzt, kann die Energie einer Feder zwischen den Punkten $\mathcal{P}_i$ und $\mathcal{P}_j$ damit wie folgt berechnet werden:

$$\mathcal{W}_{i,j} = \frac{1}{2} * \mathcal{K}_{i,j} * (\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} - l_{i,j})^2$$

wobei $l_{i,j}$ der Länge und $\mathcal{K}_{i,j}$ der Federkonstanten der Feder zwischen den beiden Punkten $\mathcal{P}_i$ und $\mathcal{P}_j$ im spannungslosen Zustand entspricht.

Die Gesamtenergie des Systems, bestehend aus allen Federn, kann dann beschrieben werden durch:

$$\mathcal{W}_{cpl} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} * \mathcal{K}_{i,j} * (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 - 2l_{i,j} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} + l_{i,j}^2)$$

Das Ziel des Spring-Embedder-Verfahrens ist, diese Gesamtenergie zu minimieren. Die Güte des Ergebnisses des Spring-Embedder-Verfahrens kann durch die im System verbleibende Restenergie abgeschätzt werden.

Kamada und Kawai stellten 1989 ein Verfahren zur Minimierung dieser Restenergie für die zweidimensionale Ebene vor [KaKa89], Kumar und Fowler übertrugen 1994 diesen Ansatz auf den dreidimensionalen Raum [KuFo94] und Simon, Steinbrückner und Lewerentz vervollständigten diesen Ansatz 2000 durch einige notwendige Peripherie (s.u.) und einige Performance-Verbesserungen [SiStLe00b]. Ein entsprechendes, im Rahmen dieser Arbeit entstandenes Werkzeug ist zudem als Public-Domain-Software verfügbar.

Die Grundidee basiert auf der Suche nach einem *lokalen Minimum* bzgl. der zurückbleibenden Restenergie. Ein lokales Minimum ist dadurch gekennzeichnet, daß jede minimale Veränderung einer Position eines Rings die Gesamtenergie erhöht, d.h. Ziel ist es, für alle Ringe, deren Größe minimal ist, so daß diese im folgenden als Punkte betrachtet werden können, Raumpositionen zu finden, für die gilt:

$$\frac{\partial \mathcal{W}}{\partial x_m} = \frac{\partial \mathcal{W}}{\partial y_m} = \frac{\partial \mathcal{W}}{\partial z_m} = 0 \ \forall 1 \le m \le n \ .$$

Die partiellen Ableitungen können jeweils aus der Formel für die Gesamtenergie gewonnen werden. Für die *x*-Richtung ergibt sich z.B. folgende Berechnung:

$$\frac{\partial \mathcal{W}}{\partial x_m} = \sum_{j=1}^{n} \mathcal{K}_{m,j} * ((x_m - x_j) - \frac{l_{m,j}(x_m - x_j)}{\sqrt{(x_m - x_j)^2 + (y_m - y_j)^2 + (z_m - z_j)^2}})$$

Für die Nullstellenberechnung der jeweiligen Ableitungen wird das *Newton-Raphson-Verfahren* eingesetzt (vgl. z.B. [Heus93], S. 406ff).

Der Spring-Embedder-Algorithmus geht nun wie folgt vor (vgl. [Chen99], S. 74f):

1. Berechnung des Punktes mit dem höchsten Potential für eine Reduktion der Gesamtenergie, d.h. die Summe der Absolutbeträge der partiellen Ableitungen für ihn ist maximal.
2. Anwendung des Newton-Raphson-Verfahrens, um diesen Punkt iterativ derart zu verschieben, daß die Summe aller Absolutbeträge der partiellen Ableitungen kleiner als ein Minimalwert ε ist.
3. Solange noch Punkte existieren mit einer Summe der Absolutbeträge der partiellen Ableitungen, die größer als der Minimalwert ε ist, setzt der Algorithmus wieder bei Punkt 1 an.

Für die Anwendung des Spring-Embeder-Algorithmus im Rahmen dieser Arbeit wird dieser, ausgehend von grundlegenden Problemen seiner Anwendung, noch um folgende Techniken erweitert, um in der Praxis zuverlässig Ergebnisse zu produzieren (vgl. [SiStLe00b]):

- Der Algorithmus liefert ausgehend von einer zufälligen Startkonstellation der Punkte nur ein System aus *lokalen* Minima. Um das Gesamtergebnis zu verbessern ist es daher sinnvoll, von neuen, zufälligen Startkonstellationen das Spring-Embedder-Verfahren erneut anzuwenden und die jeweils verbleibenden Restenergien zu vergleichen. Für typische Systeme mit über 1000 Knoten wurden in Experimenten nach 10 Iterationen Verbesserungen um mehrere Größenordnungen ermittelt.
- Der Newton-Raphson-Algorithmus terminiert nicht notwendigigerweise (vgl. [Heus93], S. 407). Eine Möglichkeit, solche divergierenden Fälle abzufangen ist der Einsatz der *Simulated-Annealing* Methode (vgl. z.B. [PTVF92], §10.9). Sie verwendet eine Analogie aus der Thermodynamik: Bei hohen Temperaturen von dann flüssigem Metall bewegen sich die einzelnen Moleküle sehr stark. Mit abnehmender Temperatur wird ebenfalls diese Bewegung eingeschränkt. Ist der Abkühlungsprozeß langsam genug, so ordnen sich die Moleküle selbst

wohlstrukturiert an und bilden am Ende reine Kristalle mit einer minimalen Restenergie; anderenfalls – wenn der Abkühlungsprozeß sehr schnell stattfindet – bilden sich sehr amorphe Strukturen mit einer deutlich höheren Restenergie. Dieses Verfahren wird auf den Newton-Raphson-Algorithmus übertragen: Zu Beginn ist seinen Werten noch eine hohe Varianz erlaubt; mit zunehmenden Iterationsschritten wird allerdings verlangt, daß die Werte kleiner werden, d.h. die Nullstelle eventuell erreicht wird. Die Simulated-Annealing-Methode fungiert in diesem Kontext folglich als obere Werteschranke, die mit zunehmenden Iterationen sinkt. Terminiert das Newton-Raphson-Verfahren schnell, spielt diese zusätzliche Technik keine Rolle. Divergiert es allerdings deutlich, übersteigen seine Werte diese obere Werteschranke und es kann entsprechend reagiert werden. Als Parameter für diese Art der Anwendung der Simulated-Annealing-Methode fließen ein a) die Startenergie, bis zu der die Werte des Nullstellenverfahrens zu Beginn reichen dürfen, b) die Anzahl der maximalen Iterationen, die für die Erreichung der Nullstellenfindung verwendet werden sollen und c) der konkrete Verlauf dieser Werteschranke zwischen der Startenergie und der Energie 0 nach den maximal zulässigen Iterationen (z.B. linearer oder logarithmischer Verlauf).

- Die Terminierung des Spring-Embedder-Verfahrens kann nicht garantiert werden, da zyklische Punkteverschiebungen möglich sind, d.h. die Optimierung eines Punkts wird durch die Optimierung von n anderen Punkten wieder zunichte gemacht. Im einfachsten Fall sind Konstellationen vorstellbar, in denen jeweils zwei Punkte oszillieren, d.h. die Schleife mit dem Abbruchkriterium nie verlassen wird. Auch für diesen Fall kann die Simulated-Annealing Methode eingesetzt werden: Dabei wird sie hierbei als obere Werteschranke für die im System verbleibende Restenergie angewendet. Mit zunehmenden Iterationen wird verlangt, daß die Restenergie minimiert wird. Ist dies nicht der Fall, kann entsprechend reagiert werden.

Hauptkomponentenanalyse

„*Die Hauptkomponentenanalyse ist eine der wichtigsten multivariaten Techniken, weil sie* [...] *eine bequeme graphische Veranschaulichung der wesentlichen Struktur eines Datensatzes ermöglicht* [...] “ ([HeHe95], S. 10). Das Ziel der Hauptkomponentenanalyse (engl.: *Principal Component Analysis*, *PCA*) ist, einen komplexen Sachverhalt, der durch eine Gesamtheit von quantitativen Merkmalen beschrieben wird, in eine einfachere Form der Darstellung zu überführen. Dies wird dadurch erreicht, daß sogenannte *Hauptkomponenten* berechnet werden, die als optimale Linearkombinationen der ursprünglichen gemessenen Variablen aufgefaßt werden. Die dadurch erreichbare Reduktion des Variablenraums auf zwei oder drei Dimensionen erlaubt dann die einfache Visualisierung der gemessenen Daten.

Das Grundprinzip der PCA soll im folgenden anhand eines Datensatzes für 10 Objekte, an denen jeweils zwei Merkmale m_1 und m_2 bestimmt wurden, veranschaulicht werden. Die Kombination der Darstellung der Meßwerte beider Merkmale in ein zweidimensionales Diagramm kann wie in Abbildung 28 geschehen. Die konkreten Merkmalswerte können durch eine einfache Projektion entlang der entsprechenden Merkmalsachse (kopierte Achse oben bzw. rechts) ausgelesen werden. Das Ziel der PCA als Technik zur Dimensionsreduzierung ist – auf dieses Beispiel angewendet – die Generierung einer eindimensionalen Darstellung mit möglichst geringem Verlust der Aussagekraft der Daten.

In Abbildung 29 ist daher eine neue Achse, die erste Hauptkomponente (PC_1) eingezeichnet: Sie paßt sich offensichtlich der Punktemenge der vermessenen Objekte unter allen eindimensionalen Unterräumen (d.h. Geraden) am besten an. Die Projektion der vermessenen Punkte auf diese Hauptkomponente stellt die eindimensionale Präsentation der Daten dar. Auch wenn in diesem Fall die erste Hauptkomponente eine sehr viel größere Streuung erzeugt als die beiden Ausgangsvariablen, so kann doch selbst eine optimale Reduktion nicht die Gesamtstreuung der Objekte präsentieren, d.h. je nach Dimension der Ausgangsdaten gehen mit der Reduzierung auf die erste Hauptkomponente Informationen verloren. Daher werden häufig weitere Hauptkomponenten gebildet, die sukzessive ein Maximum der Reststreuung erfassen. Die zugehörigen Achsen stehen dabei immer senkrecht aufeinander. Die Entscheidung, wieviele Hauptkomponenten benötigt werden, ohne die ursprünglichen Daten zu sehr zu verfälschen, basiert letztlich immer auf subjektiven Kriterien (vgl. [FaHa84], S. 678).

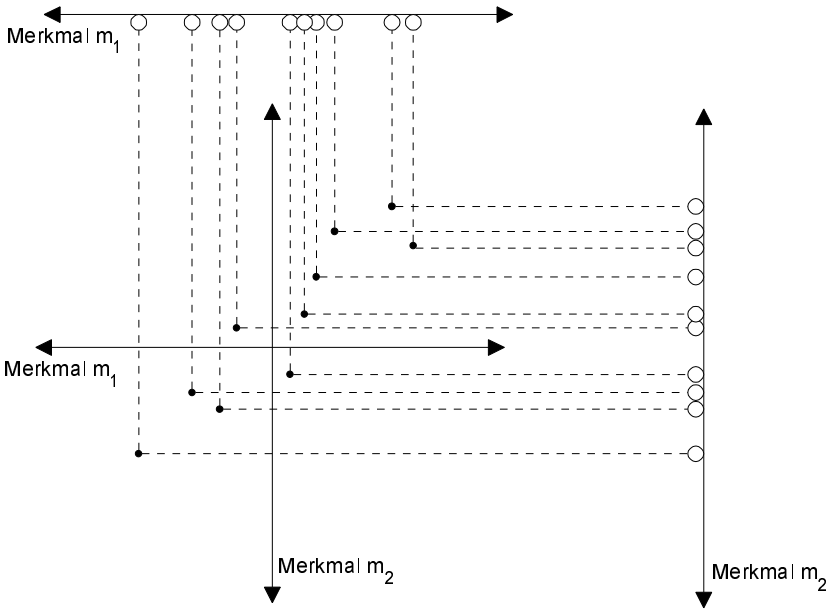


Abbildung 28: 10 Objekte mit Meßwerten für zwei Merkmale in einem zweidimensionalen Diagramm mit den Projektionen für die einzelnen Merkmale.

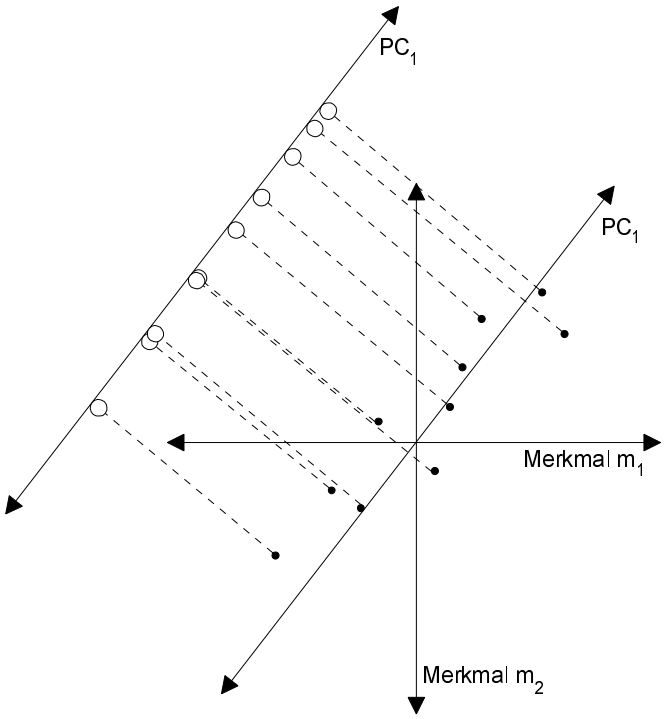


Abbildung 29: Erste Hauptkomponente für die Objekte aus Abbildung 28 mit Projektion der Meßwerte darauf

Da das Verfahren der PCA eine Standardtechnik ist und bereits in vielen statistischen Paketen wie z.B. SPSS® implementiert ist, sollen im folgenden nur grob die einzelnen Schritte erläutert werden. Für weitergehende Informationen vgl. z.B. [HeHe95], [ToStSt86] oder [Much92].

In der Regel wird vor der Anwendung der PCA eine geeignete Datenvorbehandlung durchgeführt. Diese umfaßt (vgl. [HeHe95], S. 15f)

- die *Zentrierung* der Objektpunktmenge, d.h. die Verschiebung ihres Mittelpunkts in den Koordinatenursprung, und
- die *Standardisierung* der Objektpunktmenge, d.h. alle Objekte besitzen die gleiche Standardabweichung und Streuung Eins.

Anschließend wird aus dieser Matrix die *Korrelationsmatrix* bestimmt: Sie errechnet sich für zwei Vektoren aus der Kovarianz beider Vektoren geteilt durch die multiplizierten Standardabweichungen beider Vektoren. Innerhalb der Korrelationsmatrix steht auf der Diagonalen immer der Wert Eins, da eine Variable zu sich selbst maximale (positive) Korrelation besitzt. Der numerische Kern der Hauptkomponentenanalyse besteht in der Bestimmung der p Eigenwerte und normierten Eigenvektoren dieser Korrelationsmatrix:

- Die Komponenten der normierten Eigenvektoren enthalten die gesuchten optimalen Gewichte der Ausgangsvariablen bei der Bildung der jeweiligen Hauptkomponente. So ergibt sich für das Beispiel in Abbildung 28 anhand des ersten Eigenvektors eine 1:1-Gewichtung der beiden Merkmalsachsen m_1 und m_2.
- Die Eigenwerte geben die Varianz an, die durch die entsprechende Hauptkomponente repräsentiert wird. Dabei entspricht die Summe der Varianzen der ersten i Hauptkomponenten der Summe der Varianzen der ersten i Ausgangsvariablen innerhalb der Korrelationsmatrix. Aus der Berechnung des Verhältnisses des i-ten Eigenwerts zur Summe aller anderen Eigenwerte läßt sich somit die durch die entsprechende Hauptkomponente dargestellte Varianz der ursprünglichen Daten berechnen (vgl. [ToStSt86], S. 114f). Während die Varianz innerhalb der Korrelationsmatrix für jede Variable gleich ist, „[...] *polarisiert sich dieser Beitrag bei den Hauptkomponenten in fallender Weise von sehr großer Streuung (erste Hauptkomponente) bis zu sehr kleiner Streuung (letzte Hauptkomponente)*“ ([HeHe95], S. 12).

Dieses allgemeine Vorgehen der PCA kann direkt auf eine Distanzmatrix übertragen werden: Die Distanzen einer Entität zu den jeweils anderen Entitäten werden lediglich als Merkmalsvektor aufgefaßt, d.h. bei n Elementen und der damit berechenbaren, symmetrischen n× n Distanzmatrix mit den Elementen $d_{i,j}$ ergibt sich für eine Entität e_i der Merkmalsvektor ($d_{1,i}$, $d_{2,i}$, ...$d_{n,i}$). Damit kann die Distanzmatrix wie beschrieben zentriert und standardisiert werden, um anschließend die Eigenwerte und Eigenvektoren für die Bestimmung der Hauptkomponenten zu berechnen.

Sowohl das Spring-Embedder-Verfahren als auch die Hauptkomponentenanalyse sind prinzipiell in der Lage, große Distanzmatrizen mittels einer Dimensionsreduktion in den dreidimensionalen Raum zu transformieren. Die Güte dieser Transformation kann wie folgt definiert werden:

- beim Spring-Embedder-Verfahren aus der Höhe der im System durch nicht optimal entspannte Federn zurückbleibenden Gesamtenergie und
- bei der Hauptkomponentenanalyse durch den prozentualen Anteil der dargestellten Varianz, die sich durch eine Summation der ersten drei Eigenwerte errechnen läßt.

Um sowohl die Vergleichbarkeit der beiden Verfahren untereinander als auch die Aussagestärke der errechneten Transformation bzgl. der zugrundeliegenden Skala (ordinalskaliert oder intervallskaliert, vgl. Abschnitt 6.3) bewerten zu können, ist eine detailliertere Betrachtung notwendig, die überprüft, inwieweit die Mächtigkeit sinnvoller Aussagen auf der ursprünglichen Distanzmatrix durch die Transformation reduziert wird.
Wird die merkmalbasierte Ähnlichkeitsmetrik innerhalb einer Ordinalskala verwendet (vgl. Abschnitt 6.3.1), so sind paarweise Größenvergleiche möglich. Die Mächtigkeit sinnvoller

Aussagen wird durch die Transformation in den dreidimensionalen Raum nicht reduziert, wenn jede Relation der Art $(e_i, e_k) \leq (e_l, e_m)$ genau dann im transformierten, dreidimensionalen Raum gilt, wenn sie auch innerhalb der ursprünglichen Distanzmatrix gilt. Auf Grundlage dieser Relationserhaltung wird in [SiStLe00b] ein spezielles Qualitätsmaß für dimensionsreduzierende Verfahren vorgeschlagen, das angibt, wieviel Prozent der Relationen innerhalb der ursprünglichen Distanzmatrix ebenfalls in der dimensionsreduzierten Darstellung gültig sind. Die Angabe dieses Maßes ist für produzierte Transformationen wichtig, um abschätzen zu können, mit welcher Zuverlässigkeit Aussagen aus der transformierten Darstellung abgeleitet werden können. Die Qualität einer Anwendung des Spring-Embedders auf ein System mit 1287 bzgl. der merkmalbasierten Ähnlichkeit vermessenen Entitäten wird z.B. in [SiStLe00b] mit 92,1% angegeben, d.h. nur 7.9% der innerhalb des transformierten Raums dargestellten Relationen sind falsch.
Wird die merkmalbasierte Ähnlichkeitsmetrik innerhalb einer Intervallskala verwendet, so können die dimensionsreduzierenden Techniken nicht angewendet werden, da diese Verfahren die Intervalle so stark modifizieren, daß deren Interpretation innerhalb der transformierten Darstellung nicht auf die ursprüngliche Distanzmatrix übertragen werden kann, d.h. lediglich auf den Ergebnissen der Transformation beruht.

Die Entscheidung für eines der Verfahren wird im folgenden entlang der Dimensionen *Güte*, *Performance* und *Parametrisierbarkeit* vorgenommen:

- *Güte*: Bei kleinen Systemen mit weniger als 100 Entitäten kann kein signifikanter Qualitätsunterschied der beiden Verfahren bzgl. des vorgestellten Qualitätsmaßes festgestellt werden. Der in der Literatur häufig aufgeführte Nachteil der Hauptkomponentenanalyse, kleine Distanzen würden überdurchschnittlich stark in die Berechnung einfließen (vgl. z.B. [EiWi99], S. 211) hat offenbar keine Auswirkung auf die Relationserhaltung.
- *Performance*: Die in vielen Softwarepaketen standardmäßig enthaltene Implementierung der Hauptkomponentenanalyse liefert die Ergebnisse in vertretbarer Zeit (innerhalb von *Mathematica*® auf einem normalen PC z.B. für 40 Entitäten innerhalb einiger Minuten). Für größere Datenmengen sind die Programme allerdings aufgrund der Größe der Distanzmatrix und der Probleme der Eigenwertberechnung sehr großer Matrizen häufig ungeeignet.
  Die Performance des Spring-Embedders läßt sich damit nur schwer vergleichen, da es sich hierbei um ein Optimierungsverfahren handelt, dessen Ergebnisse mit zunehmender Zeit immer besser werden. Ein wesentlicher Vorteil dieses Verfahrens bzgl. der Performance ist allerdings die Möglichkeit, das Optimierungsverfahren jederzeit abbrechen zu können, um den bis dahin erreichten Zustand zu betrachten.
- *Parametrisierbarkeit*: Das Spring-Embedder-Modell erlaubt eine hohe Flexibilität aufgrund der einfließenden Federkonstante $\mathcal{K}$, die für jede Feder explizit gesetzt werden kann. So wird z.B. in [SiStLe00b] ein Anwendungsfall beschrieben, bei dem durch Setzen unterschiedlicher Federstärken die Qualität der Transformation bzgl. der Relationserhaltung weiter optimiert werden kann. Eine Möglichkeit ist z.B., Entitäten, die eine maximale Distanz zueinander haben, mit einer schwächeren Feder zu verbinden, als Entitäten, die eine geringe Distanz zueinander haben. Die dadurch forcierte Eigenschaft, daß die Priorität des Einschwingens auf eng beieinander liegenden Entitäten liegt, unterstützt spätere, auf der Darstellung der transformierten Daten basierende Prozesse.

Für die weiteren Arbeiten wird für die Dimensionsreduktion im wesentlichen ein eigens entwickelter Spring-Embedder verwendet (vgl. [SiStLe00b]), da besonders die Kriterien Unterbrechbarkeit und Parametrisierbarkeit für die weiteren Arbeiten von Bedeutung sind.

6.5 Zusammenfassung

Auf Grundlage des aristotelischen Gesetzes der Ähnlichkeit und dessen Anwendung innerhalb der Psychologie kann ein merkmalbasiertes Ähnlichkeitskonzept hergeleitet werden, das als Kantenattributierung des im letzten Kapitels vorgestellten Graphenmodells der Softwareproduktvermessung eine wirkliche Erweiterung bisheriger Meßansätze erlaubt. Das zugrundeliegende Verständnis von Ähnlichkeit beruht dabei auf der Annahme, daß zwei Entitäten um so ähnlicher sind, je mehr gemeinsame Merkmale sie im Verhältnis zur Vereinigungsmenge ihrer Merkmale besitzen. Mittels der bewußten Selektion der für das jeweilige Ähnlichkeitsverständnis relevanten Merkmale kann ein generisches Distanzmaß erstellt werden, das – verstärkt durch dessen metrischen Charakters – deutlich von den allgemeinen Maßen zu unterscheiden ist und für viele unterschiedliche Sichtweisen auf ein Softwaresystem instanziiert werden kann. Die Struktur des zugrundeliegenden Produktmodells und die möglichen Anwendungen der Metrikwerte bleiben dabei fix, so daß unabhängig von der konkreten Instanziierung des generischen Distanzmaßes sowohl eine ordinalskalierte Kantenvermessung als auch eine intervallskalierte Knotenvermessung ermöglicht wird. Neben den Vorteilen der ausgeprägten Intuition des Menschen gegenüber dem Distanzkonzept und der damit intuitiven Verständlichkeit der Metrikdaten sind vor allen Dingen die aussagekräftige und auch für große Systeme skalierende Visualisierung und Clusteranalyse ein weiterer Vorteil dieser Art von Daten. Innerhalb typischer Anwendungsfälle der erzeugten Metrikwerte spielt für die Visualisierung im folgenden besonders die Erstellung dreidimensionaler, virtueller Informationsräume mittels der Spring-Embedder-Technik eine herausragende Rolle; bei den Clusteranalyse-Verfahren werden besonders die hierarchisch agglomerativen Verfahren verwendet.
Im folgenden Kapitel wird das Distanzmaß für eine intuitivere Herangehensweise an das bekannte Qualitätskriterium Kohäsion von Softwaresystemen verwendet, da dies hilft, bekannte Kohäsionsmaße zu validieren oder zu falsifizieren, verschiedene Kohäsionsarten zu klassifizieren und gegebenenfalls neue, angepaßtere Formen zu instanziieren.

„So kann sich Gravitation
durch ihre eigene Wirkung
in Kohäsion verwandeln,
indem sie die Teilchen aus merklicher Entfernung
zur Berührungsnähe bringt."
(G. Th. Fechner: „*Über das Causalgesetz*")

7 Distanzmaßbasierte Kohäsion

Mit dem im letzten Kapitel vorgestellten generischen und metrischen Distanzmaß ist es möglich, Distanzen zwischen verschiedenen Elementen auf Grundlage einer für die jeweilige Betrachtung relevanten Merkmalsträgermenge zu bestimmen. Während diese Distanz innerhalb des Spring-Embedders als geometrische Distanz und innerhalb der Clusteranalyse als Kriterium für das konstruktive Erstellen von Gruppierungen herangezogen wird, kann sie allgemein für die Untersuchung verwendet werden, ob und inwieweit einige betrachtete Elemente bzgl. einer speziellen Sichtweise zusammengehören. Bezogen auf den Fokus der Arbeit, nämlich die meßwertbasierte Analyse von Softwaresystemen, läßt sich die Distanzbestimmung dort in den Kontext der Bestimmung von *Kohäsion*, einem wichtigen Qualitätsziel von Software, einordnen. Während der Begriff Kohäsion, der seinen hermeneutischen Ursprung im lateinischen *cohaerere*, *cohaesum* hat und soviel wie „zusammenhängen" bedeutet, allgemein eher der Physik zuzuordnen ist und dort „[...] *the intermolecular attractive force acting between two adjacent portions of a substance* [...] *that holds a piece of matter together*" (Encyclopaedia Britannica, unter cohesion) bezeichnet, haben bereits 1974 Stevens, Myers und Constantine das Qualitätsziel Kohäsion innerhalb ihres *strukturierten Designs* für Softwaresysteme definiert und gefordert ([StMyCo74]). Bzgl. der Anordnungen der Anweisungen in Modulen wird von ihnen empfohlen, „[...] *that statements within any module are closely related and any pair of statements in two different modules are minimally related* " ([Myer76], S. 89).
In diesen Arbeiten wurde für die Qualitätseigenschaft Kohäsion eine Abstufung in mehrere geordnete Kohäsionsstufen vorgenommen, die von nicht kohäsiv bis maximal kohäsiv reichen (vgl. Abschnitt 7.1). Während der Fokus dieser Ausprägungen nicht in der Meßbarkeit lag, wurde in den folgenden Jahren eine Vielzahl von Softwaremaßen vorgestellt, die Kohäsion eben meßbar machen sollten. Diese Anzahl nahm noch einmal deutlich mit dem Aufkommen des objektorientierten Programmierparadigmas zu, da mit der Einführung des Konzepts der Klasse die Fragestellung aufkam, welche Methoden und Attribute zusammen eine gute Klasse ausmachen.
Die Vielzahl vorgestellter Kohäsionsmaße, deren unterschiedlichste Produktmodelle, deren unterschiedlichste Meßanwendungen und deren teilweise sehr fragwürdige Interpretationen der Meßergebnisse ließen bis heute nur selten eine überzeugende, automatische Kohäsionsvermessung zu: „*In the past measures for cohesion mostly were neglected*" ([Zuse98], S. 556).

Mittels der distanzmaßbasierten Softwarevermessung ist es nun möglich, die Kohäsionsbestimmung als konkrete Instanziierung des generischen Distanzmaßes (und einer speziellen Bewertungsfunktion, s.u.) aufzufassen. Daraus ergeben sich folgende Vorteile:

1. Die Kohäsionsbestimmung kann mittels einer einheitlichen Nomenklatur erläutert werden und reduziert damit deutlich das Maß-Definitions- und das Maß-Selektions-Problem (vgl. Kapitel 3.6.2). Die Notwendigkeit einer einheitlichen Begriffswelt für die Kohäsionsbetrachtung wurde bereits in [BrDaWü97] gezeigt. Dort beschränkt sich die

Vereinheitlichung allerdings auf die Beschreibung rein syntaktischer Details, die für die Erstellung der Produktmodelle notwendig sind. Das jeweils einem Maß zugrundeliegende empirische Relationensystem und dessen Validität bleiben dort allerdings unberücksichtigt.

2. Aufgrund der ausgeprägten Intuition des generischen Kohäsionskonzepts und der Einfachheit dessen Instanziierung sind die Kohäsionsmaße einfacher zu validieren bzw. können die innerhalb des empirischen Relationensystems erlaubten Aussagen leichter hergeleitet werden (vgl. Wert-Interpretations-Problem). Dies erlaubt u.a. für viele Maße eine deutliche Reduktion ihrer Aussagekraft (z.B. durch das Beschränken auf ausschließlich klassifizierende anstelle ordinaler Aussagen).
3. Es wird deutlich, warum die für viele bisherige Kohäsionsmaße notwendige *Bewertungsfunktion*, die verschiedene, intern vorgenommene Kantenvermessungen zwischen Elementen in die sie jeweils besitzende Entität als Knotenvermessung überführt (z.B. durch die Bildung des Mittelwerts), die Interpretation der Meßwerte aufgrund ihrer Wertenivellierung verschlechtert. An die Stelle einer knotenorientierten Aussage, eine Entität besitze schlechte Kohäsion, tritt damit die für konstruktive Restrukturierungen sehr viel wertvollere Aussage, welche Elemente dieser Entität eine hohe Distanz besitzen und damit die Gesamtkohäsion der Entität reduzieren.
4. Die verschiedenen vorgestellten Techniken der distanzmaßbasierten Vermessung wie Visualisierung oder Clusteranalyse (vgl. Kapitel 6.4) lassen sich als zusätzliche Interpretation bisheriger Kohäsionsmaße automatisch erstellen.

In Abschnitt 7.1 wird versucht, einige typische Kohäsionsmaße derart mittels des generischen Distanzmaßes zu realisieren, daß diese vier Vorteile größtmögliches Gewicht erhalten.
Ein weiterer wichtiger Vorteil dieser Herangehensweise ist die in die Kohäsionsvermessung integrierte Betrachtung von Kopplung und damit verbunden die Lösung des Kopplungs-Kohäsions-Problems: An die Stelle des größtenteils separaten Vermessens von Kopplung und Kohäsion tritt die *durch Kopplung begründete Kohäsion*, d.h. die Art der betrachteten Beziehungen, die ein Zusammengehören von Entitäten begründet, ist durch Kopplung gegeben. Um dieses Konzept der Kohäsionsbestimmung maximal ausnutzen zu können, wird in Abschnitt 7.2 ein Überblick über existierende Sichtweisen von Kopplung gegeben.
Mit der Übersicht über existierende Betrachtungsweisen von Kohäsion und Kopplung sind genügend Aspekte für ein System zusammengetragen, um alle Parameter für die Anwendung des generischen Distanzmaßes zu identifizieren. In Abschnitt 7.3. wird daher ein Prozeß zur kopplungbasierten Kohäsionsbestimmung vorgestellt. Da die für Kohäsion betrachteten Beziehungen allerdings nicht auf Kopplung beschränkt sein müssen, wird in Abschnitt 7.4 eine weitere Abstraktion des Prozesses vorgestellt, das alle Kohäsionsarten subsumiert. Konkrete Anwendungen beider Prozesse sind in Kapitel 8 und 9 beschrieben.

7.1 Kohäsion

Der Vielzahl von unterschiedlichen Interpretationen für den Begriff der Kohäsion und den damit verbundenen, unterschiedlichen Kohäsionsarten kann folgende, allgemeine Definition für Kohäsion innerhalb der Softwaretechnik zugrunde gelegt werden:

> Kohäsion betrachtet den Grad der Zusammengehörigkeit der in einer zu betrachtenden Entität enthaltenen Elemente.

Definition 33: Kohäsion (vgl. [BiKa98], S. 111)

Die verschiedenen Kohäsionsbetrachtungen und deren spezifische Maße unterscheiden sich entsprechend dieser Definition in den folgenden drei Dimensionen:

- *Art der zu betrachtenden Entität*: Hierbei geht es um die Art derjenigen Entitäten, für die die Kohäsionsbetrachtung durchgeführt werden soll. Beispiele für solche Arten von Entitäten sind z.B. Klassen (Klassenkohäsion), Funktionen (Funktionskohäsion) oder Module (Modulkohäsion).
- *Art der enthaltenen Elemente innerhalb der betrachteten Entität*: Die Kohäsionsbetrachtung einer Entität verlangt die Betrachtung darin vorkommender Elemente, d.h. zwischen der Entität und ihren Elementen muß eine „*Ist-Teil-Von*"-Relation vorliegen (vgl. Kapitel 5.1.3). In der Regel wird als Art der Elemente einer Entität dabei die direkt nächste Ebene einer Enthaltensein-Dekomposition der Entität verwendet. Bei einer mehrfachen Verfeinerung der Entität in ihre Elemente, die im Folgeschritt ihrerseits wieder als Entität aufgefaßt werden und damit wiederum Elemente besitzen, können ausgehend von einer Entität mehrere Arten von Elementen für die Kohäsionsbetrachtung einer Entität verwendet werden. Beispiele für solche Arten von Elementen einer betrachteten Entität sind Methoden für Klassenkohäsion, Variablen für Funktionskohäsion oder Funktionen für Modulkohäsion.
- *Art der Zusammengehörigkeit*: Hierbei geht es darum, bzgl. welcher Kriterien die Elemente einer Entität zusammengehören. Beispiele für eine Zusammengehörigkeit sind z.B. für Methoden die gemeinsam benutzten Attribute oder für Variablen die Änderungsabhängigkeit.

Anhand dieser Definition wird noch einmal die Art der für die Kohäsionsbestimmung herangezogenen Produktmodelle deutlich (vgl. Kapitel 5.1.3): Jede zu vermessende Entität besitzt darin Elemente, deren Zusammengehörigkeit für den Kohäsionsgrad der sie beinhaltenden Entität verwendet wird.
Die oben aufgeführte Möglichkeit, bisherige Kohäsionsmaße als konkrete Instanziierungen des generischen Distanzmaßes aufzufassen, kann wie folgt beschrieben werden: Mittels einer konkreten Merkmalsträgermenge kann das Distanzmaß *dist* auf alle Elemente e_i der bzgl. Kohäsion zu vermessenden Entität E angewendet werden. Die damit ermittelten Werte werden anschließend mittels einer geeigneten, vom jeweiligen Kohäsionsmaß abhängigen Bewertungsfunktion ψ in einen einzigen Wert für die Entität überführt (häufig nach vorheriger Skalierung der ermittelten Distanzwerte). Diese Bewertungsfunktion „bewertet" dabei lediglich auf rein numerischer Basis die ermittelten Distanzen (z.B. durch Summation aller Distanzen).
Da E, e_i und *dist* untereinander nicht unabhängig sind, kann folgende grobe Klassifizierung existierender Kohäsionsmaße und der dahinter stehenden Kohäsionsbetrachtung identifiziert werden:
- *Funktionale Kohäsion*, bei der die Entitäten und deren Elemente einer funktionalen Modellierung und Implementierung des Problems entnommen sind und die Zusammengehörigkeit sich auf die funktionale Zusammengehörigkeit konzentriert, und
- *abstrakte Kohäsion*, bei der die Entitäten und deren Elemente einer objektorientierten Modellierung und Implementierung des Problems mittels abstrakter Datentypen entnommen sind und die Zusammengehörigkeit sich eher auf die Zusammengehörigkeit bzgl. der Daten konzentriert (vgl. [FePf96], S. 312; [BiOt94], S. 645).

Aufgrund der mit diesen unterschiedlichen Kohäsionsbetrachtungen verbundenen unterschiedlichen Entitäten mit unterschiedlichen Merkmalsmengen, die für die jeweilige Kohäsionsbestimmung verwendet werden (vgl. auch Kapitel 8), sollen im folgenden beide Formen separat behandelt werden. Für ein System können allerdings beide Kohäsionsbetrachtungen relevant sein: So kann eine Klasse bzgl. der abstrakten Kohäsion untersucht werden, d.h. inwieweit stellt die Klasse die Implementierung eines abstrakten Datentyps dar, während ihre Methoden jeweils bzgl. der funktionalen Kohäsion untersucht werden sollen, d.h. inwieweit stellt eine Methode eine zusammengehörende Funktionalität dar.

7.1.1 Funktionale Kohäsion

Innerhalb der funktionalen Kohäsion liegt der Schwerpunkt der Betrachtung darauf, inwieweit die Elemente eines Moduls oder einer Funktion zur Realisierung einer einzigen funktionalen Anforderung benötigt werden, d.h. ob die gewählte Aufteilung des Gesamtproblems in Teilprobleme entsprechend einer funktionalen Dekomposition vorgenommen wurde. Als Gegenbeispiel einer derart überlegten Struktur beschreibt Pressman exemplarisch folgendes, in den Anfängen der strukturierten Programmierung durchgeführtes Verfahren zur Aufteilung in Teilfunktionen: Ein bestehendes Fortran-Programm mit 20.000 Programmzeilen wurde ausgedruckt und alle 75 Zeilen mit dem Lineal ein horizontaler Strich eingefügt. Jedes Programmfragment zwischen zwei Strichen stellte anschließend eine Funktion dar ([Pres97], S. 358)!

Um derartigen Auswüchsen entgegentreten zu können, haben Stevens, Myers und Constantine bereits 1974 für das Qualitätsziel Kohäsion innerhalb ihres *strukturierten Designs* ein *Kohäsionsspektrum* vorgestellt, das die unterschiedlichen Grade vorgefundener funktionaler Kohäsion ordinal bewertet [StMyCo74]. Die einzelnen geordneten sieben Stufen[20], über die bzgl. ihrer jeweiligen Intervalle explizit keine Aussagen gemacht werden sollen („*The scale is not linear* ", [ebd.], S. 237), können dabei wie in Abbildung 30 dargestellt angeordnet werden.

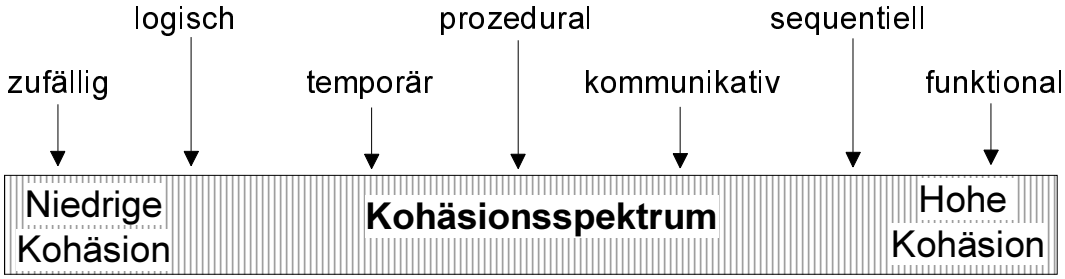


Abbildung 30: Kohäsionsspektrum (vgl. [Press97], S. 358)

Die einzelnen Stufen bedeuten dabei:

1. *Zufällige funktionale Kohäsion (coincidental)*: Hierbei besitzen die Elemente einer Funktion keinerlei funktionale Zusammengehörigkeit. Funktionen, die mittels der oben wiedergegebenen Aufteilung via fester Zeilenanzahl in Teilfunktionen aufgeteilt werden, besitzen üblicherweise lediglich zufällige Kohäsion.
2. *Logische funktionale Kohäsion (logical)*: Hierbei gehören die Elemente einer Funktion logisch zusammen, d.h. sie decken einen gemeinsamen, nicht funktionalen Aspekt des Problembereichs ab. Typische Formen solcher Aspekte für funktionale Kohäsion sind datenstromorientierte Gruppierungen wie INPUT oder OUTPUT-Funktionen oder auch ressourcenorientierte Gruppierungen wie HARDDISK oder FLOPPY-Funktionen.
3. *Temporäre funktionale Kohäsion (temporal)*: Hierbei besitzen die Elemente einer Funktion logische funktionale Kohäsion, die einzelnen Elemente werden allerdings darüber hinaus nicht zeitlich separat verwendet, sondern innerhalb bestimmter Zeitfenster. Typische Beispiele solcher Funktionen sind Initialisierungsfunktionen, Terminierungsfunktionen oder auch Aufräumfunktionen.
4. *Prozedurale funktionale Kohäsion (procedural)*: Hierbei sind die Elemente durch einen Kontrollfluß miteinander verbunden. Typische Beispiele solcher Funktionen sind

[20] Während Stevens, Myers und Constantine 1974 nur sechs Stufen aufzeigten, hat Myers in seinem Buch „*Software Reliability*" noch eine siebente Stufe – die prozedurale funktionale Kohäsion – eingefügt [Myer76]. Diese Unterscheidung wird i.d.R. heute ignoriert und nur noch alle sieben Stufen betrachtet (vgl. z.B. [BrDaWü97], [Zuse98], [FePf96]).

Implementierungen von problemorientierten Abläufen wie z.B. Eingabe→Verarbeitung→Ausgabe (vgl. z.B. „*EVA-Prinzip*“ in [Dude88]).

5. *Kommunikative funktionale Kohäsion (communicational)*: Hierbei besitzen die Elemente einer Funktion prozedurale funktionale Kohäsion, die einzelnen Elemente gehören allerdings zusätzlich bzgl. ihrer Menge verwendeter Aus- und Eingabevariablen zusammen. Typische Formen sind problemorientierte Abläufe bzgl. ähnlicher Datenbestände wie z.B. das Drucken mit anschließendem Löschen eines Datenbestands.
6. *Sequentielle funktionale Kohäsion (sequential)*: Hierbei sind die Elemente einer Funktion Teile eines gemeinsamen Datenflusses, d.h. die Ausgabe des einen Elements dient zur Eingabe eines anderen Elements. Typische Beispiele sind Datenstrukturkapselungen, bei denen die innere Struktur von Daten verdeckt wird und lediglich durch in bestimmter Reihenfolge auszuführende Funktionsaufrufe verwendet werden kann (z.B. Listenoperationen, wobei der innere Aufbau der Liste verdeckt wird).
7. *Funktionale Kohäsion (functional)*: Hierbei dienen alle Elemente einer Funktion lediglich der Realisierung einer speziellen Funktion des Problembereichs.

In einigen Arbeiten wird der mögliche Zusammenhang zwischen abstrakter und funktionaler Kohäsion (vgl. Abschnitt 7.1) durch eine zusätzliche achte Kohäsionsstufe - die *informationelle Kohäsion* - erweitert (z.B. [Fair85], Kapitel 5.2.1). Hierbei wird angenommen, daß eine hohe funktionale Kohäsion verschiedener Softwareelemente notwendige Voraussetzung für deren abstrakte Kohäsion auf der nächst höheren Enthaltenseinsstufe darstellt: „*Each routine in the module exhibits functional cohesion*“ ([Fair87], S. 150).

Die Kohäsion einer Funktion ist in allen Fällen definiert als die höchste Kohäsionsstufe, die sie erfüllt. Die unterschiedlichen Ausprägungen von Zusammengehörigkeit der Elemente einer Funktion können nicht automatisch identifiziert werden. Statt dessen wird ein heuristisches, auf die Mitarbeit des Entwicklers der Funktion angewiesenes Verfahren vorgestellt (s. Kapitel 3.2.3).
Wesentliche Arbeiten zur automatischen Bestimmung der funktionalen Kohäsion wurden von Biemann, Ott und Thuss durchgeführt (z.B. [BiOt92], [OtTh93], [BiOt94]); die dort beschriebenen Kohäsionsmaße sind typisch für die Bestimmung funktionaler Kohäsion, gut dokumentiert und darüber hinaus extern validiert.
Die dort vorgestellten Maße lassen sich entsprechend der oben vorgestellten Kohäsionsdimensionen wie folgt beschreiben:

- *Art der zu betrachtenden Entität*: Betrachtet wird die Kohäsion von Funktionen bzw. Prozeduren („*Functional cohesion is actually an attribute of individual procedures or functions* [...]“ [BiOt94], S. 645).
- *Art der Elemente der betrachteten Entität*: Als Elemente der vermessenen Entität, deren Zusammengehörigkeit das Kohäsionsmaß begründet, werden diejenigen Daten-Token einer Funktion verwendet, die für mindestens ein Funktionsergebnis, d.h. Ausgabeparameter, modifizierte globale Variablen und modifizierte globale Dateien, verwendet werden. Diese Daten-Token wurden von Emerson eingeführt und dort als *variable-referent executable statements* (VRES) bezeichnet [Emer84].
- *Art der Zusammengehörigkeit der Elemente*: Die Zusammengehörigkeit der Elemente wird aufgrund der Daten-Slices der Ausgabeparameter durchgeführt (s. Kapitel 3.2.3): Zwei Daten-Token gehören um so mehr zusammen, in je mehr Daten-Slices der Funktionsergebnisse sie vorkommen. Daten-Token, die in mindestens zwei Daten-Slices vorkommen, werden als *Glue-Token*, Daten-Token, die in allen Daten-Slices vorkommen, als *Super-Glue-Token* bezeichnet [BiOt94].

Das hinter dieser Instanziierung liegende Grundverständnis von Kohäsion besteht aus einer Analyse der Verhältnisse von Glue- bzw. Super-Glue-Token zu den insgesamt vorkommenden

Daten-Token. Der Extremwert der zufälligen Kohäsion, d.h. es werden mehrere Funktionen berechnet, die untereinander keinerlei Beziehungen besitzen, läßt sich in diesem Modell dadurch charakterisieren, daß es keine Glue- und Super-Glue-Token gibt. Im Gegensatz dazu bedeutet die funktionale Kohäsion, daß jedes Daten-Token zugleich ein Super-Glue-Token ist.
Für eine graduelle Bestimmung der Ausprägung von Kohäsion werden drei Kohäsions-Maße für eine Funktion *f* vorgestellt:

- $SFC_{(strong\ functional\ cohesion)}(f) := \frac{|Super\_Glue\_Token(f)|}{|Daten\_Token(f)|}$
- $WFC_{(weak\ functional\ cohesion)}(f) := \frac{|Glue\_Token(f)|}{|Daten\_Token(f)|}$
- $Adhesiveness\ (f) := \frac{\sum_{t \in Daten\_Token(f)} |\{s \in Daten\_Slices(f) : t \in s\}|}{|Daten\_Token(f)| * |Daten\_Slices(f)|}$

Diese drei Maße können unmittelbar in eine konkrete Instanziierung des generischen Distanzmaßes überführt werden:

- Die Entitäten, deren jeweilige Distanz zueinander bestimmt wird, sind durch alle vorkommenden Daten-Token gegeben.
- Als Merkmalsträgermenge werden die einzelnen Slices verwendet, d.h. die für diesen Kontext relevanten Merkmale einer Entität sind diejenigen Slices, in denen das Daten-Token vorkommt.

Zusätzlich wird noch eine künstliche Entität e_{cpl} benötigt, die alle vorkommenden Slices als Merkmale besitzt, d.h. e_{cpl} ist ein Super-Glue-Token. Damit sind folgende Neuformulierungen der Kohäsionsmaße möglich:

- WFC: Ein Daten-Token e_i ist ein Glue-Token, wenn gilt:
  $dist(e_i, e_{cpl}) < \frac{|Daten\_Slices(f)| - 1}{|Daten\_Slices(f)|}$.
  Gilt $dist(e_i, e_{cpl}) = \frac{|Daten\_Slices(f)| - 1}{|Daten\_Slices(f)|}$, so besitzt e_i nur einen Slice als Merkmal.

  Damit gilt: $WFC(f) := \frac{\sum_{t \in Daten\_Token(f)} \xi(dist(t, e_{cpl}))}{|Daten\_Token(f)|}$.

  Dabei skaliert ξ die Distanz jeweils derart, daß

  $\xi(x)=1$, wenn $x < \frac{|Daten\_Slices(f)| - 1}{|Daten\_Slices(f)|}$ und

  $\xi(x)=0$ sonst.
- SFC: Ein Daten-Token e_i ist ein Super-Glue-Token, wenn dessen Distanz zu e_{cpl} Null ist. Gilt $dist(e_i, e_{cpl})>0$ so besitzt e_{cpl} einen Slice als Merkmal, der nicht Merkmal von e_i ist.

  Damit gilt: $SFC(f) := \frac{\sum_{t \in Daten\_Token(f)} \vartheta(dist(t, e_{cpl}))}{|Daten\_Token(f)|}$.

Dabei skaliert ϑ die Distanz jeweils derart, daß
$\vartheta(x)=1$, wenn $x=0$ und
$\vartheta(x)=0$ sonst.

- Adhesiveness: Die Anzahl der Slices, zu denen ein Daten-Token e_i gehört, ist umgekehrt proportional zur Distanz dist(e_i, e_{cpl}). Um die Summe der Daten-Slices aller Daten-Token nachzubilden, muß die Summe der Distanzen anschließend von der Anzahl der Daten-Token subtrahiert werden. Hierdurch werden die Distanzen auf Ähnlichkeiten zurückgeführt.

Damit gilt: $$Adhesiveness(f) := \frac{|Daten\_Token(f)| - \sum_{t \in Daten\_Token(f)} dist(t, e_{cpl})}{|Daten\_Token(f)| * |Daten\_Slices(f)|}.$$

Aus diesen Neuformulierungen der Kohäsionsmaße mittels des generischen Distanzmaßes lassen sich direkt folgende Aussagen ableiten:

- Bei allen drei Kohäsionsmaßen wird Kohäsion auf Grundlage der durch in gemeinsamen Slices gegebenen Ähnlichkeit zwischen Daten-Token ermittelt. Die dadurch betrachtbaren Kohäsionsstufen umfassen alle Stufen oberhalb (und inklusive) der prozeduralen funktionalen Kohäsion, bei der die kontrollflußmodifizierenden Anweisungen als Daten-Token betrachtet werden. Eine weitere Abstufung ist allerdings nicht möglich, da nicht zwischen kontrollflußmodifizierenden Daten-Token und gemeinsamen Aus- und Eingabevariablen unterschieden wird.
- Jeder Slice und jedes Daten-Token besitzt denselben Einfluß auf die Kohäsionsbestimmung der sie enthaltenden Entität. Es wird weder zwischen Ausgabeparametern, modifizierten globalen Variablen und modifizierten Dateien unterschieden, noch werden die unterschiedlichen Arten von Daten-Token (z.B. lesend vs. schreibend) separat betrachtet.
- Bzgl. SFC und WFC gilt grundsätzlich: $SFC(f) \leq WFC(f)$. Dies folgt direkt aus einem Vergleich von ξ und ϑ.
- Das durch das Distanzmaß erzeugte Kontinuum im Intervall [0..1] wird durch die jeweiligen Skalierungsfunktionen ξ und ϑ auf die beiden Werte 0 bzw. 1 diskretisiert. Der damit verbundene Informationsverlust ist offensichtlich: Es ist für das WFC-Maß irrelevant, ob ein Daten-Token zu zwei oder zu allen Daten-Slices gehört; eine ordinale Abstufung findet nur zwischen den beiden diskreten Situationen „*ist in mehr als einem Slice vorhanden*“ und „*ist in genau einem Slice vorhanden* “ statt. Umgekehrt ist es für das SFC-Maß irrelevant, ob ein Daten-Token zu nur einem Slice oder zu (|Daten_Slices(f)|-1) Slices gehört; eine ordinale Abstufung findet nur zwischen den beiden diskreten Situationen „*ist in allen Slices vorhanden*“ und „*ist nicht in allen Slices vorhanden*“. Beide Maße erzeugen damit zwar eine Ordinalskala, allerdings ist ihre Aussagekraft künstlich reduziert, d.h. zwischen einem Programm, in dem jedes Daten-Token zu (|Daten_Slices(f)|-1) Slices gehört und einem Programm, in dem jedes Daten-Token zu zwei Slices gehört wird weder beim SFC- noch beim WFC-Maß eine ordinale Abstufung vorgenommen.
- Die Summenfunktion Σ, die als Bewertungsfunktion alle (skalierten) Einzeldistanzen in einen Gesamtkohäsionswert summiert, führt zu einem Informationsverlust der Ergebnisse: Die Summierung beliebiger Werte läßt nur noch wenig Rückschlüsse über die einzelnen Operanden zu. Dieser „Schmiereffekt“ gilt besonders für das Adhesiveness-Maß, da hier keine Diskretisierung der Werte stattfindet, sondern Werte aus dem gesamten Kontinuum des Intervalls [0..1] aufaddiert werden.
- Kein Maß erlaubt eine Aussage darüber, welche konstruktiven Schritte für eine Verbesserung der Kohäsion vorgenommen werden können. Dies ist aber für einen Großteil der von Maßen unterstützten Prozesse äußerst relevant (vgl. Kapitel 3.3).

Neben diesen aufgrund der einheitlichen Implementierungstechnik einfach zu identifizierenden Aussagen ermöglicht die distanzbasierte Kohäsionsbetrachtung unter Beibehaltung des für SFC, WFC und Adhesiveness verwendeten Kohäsionskonzepts die Anwendung der Standardtechniken Visualisierung und Clusteranalyse. Das folgende Beispiel, dessen Quelltext bereits in [OtTh89] für eine auf der Technik des Slicing basierenden Kohäsionsbetrachtung verwendet wurde, soll dies verdeutlichen:

Funktionscode	Identifizierte Daten-Token

`procedure SumAndProduct (N: integer;`
`  var SumN: integer;`
`  var SumSqrN: integer;`
`  var ProdN: integer);`	

`var I: integer;`	
`begin`	
`SumN:=0;`	$SumN_1$
`for I:=1 to N do`	I_1, N_1
`  SumN:=SumN+I;`	$SumN_2$, $SumN_3$, I_2
`SumSqrN:=0;`	$SumSqrN_1$
`for I:=1 to N do`	I_3, N_2
`  SumSqrN:=SumSqrN + I*I;`	$SumSqrN_2$, $SumSqrN_3$, I_4, I_5
`ProdN:=1;`	$ProdN_1$
`for I:=1 to N do`	I_6, N_3
`  ProdN:=ProdN*I`	$ProdN_2$, $ProdN_3$, I_7
`end`	

Die einzelnen Slices für die einzelnen Ausgabevariablen sind:

- Slice(SumN) := $\{SumN_1, I_1, N_1, SumN_2, SumN_3, I_2\}$
- Slice(SumSqrN) := $\{SumSqrN_1, I_3, N_2, SumSqrN_2, SumSqrN_3, I_4, I_5\}$
- Slice(ProdN) := $\{ProdN_1, I_6, N_3, ProdN_2, ProdN_3, I_7\}$

Die einzelnen Meßwerte für die drei Kohäsionsmaße sind:

- SFC(SumAndProduct)=0, da kein Token in allen Slices vorkommt,
- WFC(SumAndProduct)=0, da kein Token in mindestens zwei Slices vorkommt und
- Adhesiveness(SumAndProduct)= $19/(3*19) \approx 0{,}33$.

Diese bereits in [OtTh89] präsentierten „Ergebnisse“ können durch die Anwendung des Distanzmaßes (vgl. oben) sehr viel detaillierter analysiert werden: Für diesen Zweck werden die folgenden, für diesen Kontext relevanten Merkmalsmengen für die einzelnen Daten-Token ermittelt:

- $p(SumN_1) \cap \mathbb{B} = p(I_1) \cap \mathbb{B} = p(N_1) \cap \mathbb{B} = p(SumN_2) \cap \mathbb{B} = p(SumN_3) \cap \mathbb{B} = p(I_2) \cap \mathbb{B} = \{SumN_1, I_1, N_1, SumN_2, SumN_3, I_2\}$ = Slice(SumN)
- $p(SumSqrN_1) \cap \mathbb{B} = p(I_3) \cap \mathbb{B} = p(N_2) \cap \mathbb{B} = p(SumSqrN_2) \cap \mathbb{B} = p(SumSqrN_3) \cap \mathbb{B} = p(I_4) \cap \mathbb{B} = p(I_5) \cap \mathbb{B} = \{SumSqrN_1, I_3, N_2, SumSqrN_2, SumSqrN_3, I_4, I_5\}$ = Slice(SumSqrN)
- $p(ProdN_1) \cap \mathbb{B} = p(I_6) \cap \mathbb{B} = p(N_3) \cap \mathbb{B} = p(ProdN_2) \cap \mathbb{B} = p(ProdN_3) \cap \mathbb{B} = p(I_7) \cap \mathbb{B} = \{ProdN_1, I_6, N_3, ProdN_2, ProdN_3, I_7\}$ = Slice(ProdN)

Werden nun für alle Daten-Token die Distanzen zueinander berechnet, kann eine 19×19 Matrix mit den Distanzen erstellt werden. Deren Visualisierung mit einem zweidimensionalen Spring-Embedder (vgl. Kapitel 6.4.2) kann eine Visualisierung wie in Abbildung 31 dargestellt liefern. Dasselbe Verfahren angewandt auf eine leicht modifizierte Prozedur `SumAndProduct2`, in der z.B. das Token N_3 auch im Slice(SumN) vorkommt (indem die dritte Schleife z.B. nicht bis N sondern bis SumN läuft), kann zu einer Abbildung führen, wie sie in Abbildung 32 dargestellt ist.

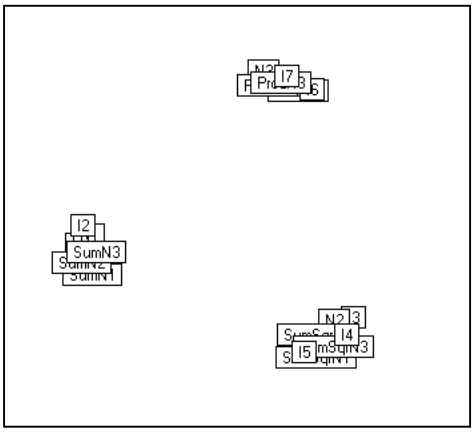


Abbildung 31: 2D-Visualisierung der Slicing-Distanzen der Prozedur `SumAndProduct`

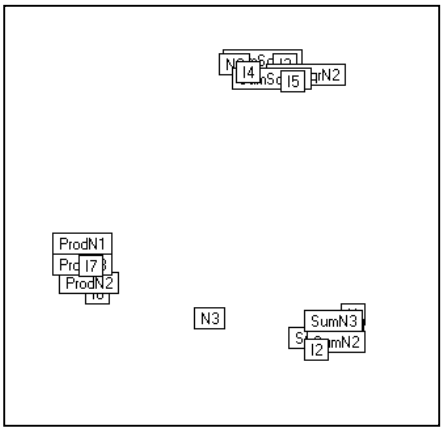


Abbildung 32: 2D-Visualisierung der Slicing-Distanzen der modifizierten Prozedur SumAndProduct2

Der Hauptvorteil dieser neuen, zusätzlichen Visualisierungen gegenüber den bloßen Kohäsionswerten (vgl. [OtTh89]) liegt in der damit neuen Möglichkeit der Ableitung konstruktiver Schritte und einer erweiterten Möglichkeit der Analyse: So ist einem schlechten Kohäsionswert der Maße SFC, WFC oder Adhesiveness nicht entnehmbar, wieviele unterschiedliche Slices in der Funktion vorkommen, ob und wenn ja welche eine Minimalkohäsion in Form wenigstens eines Glue-Tokens besitzen, wie potentielle Restrukturierungen aussähen und mit welchem Aufwand sie durchzuführen wären. Im Gegensatz dazu erlaubt Abbildung 31 das sofortige Erkennen dreier bzgl. des Slicingkriteriums vollkommen unkohäsiver Gruppen von Daten-Token. Es ist darüber hinaus klar, daß eine Restrukturierung einfach und in folgender Weise vorzunehmen ist: Nach Auslagerung der drei Gruppen von Daten-Token in separate Funktionen müssen nur noch die Clients der ehemaligen Funktion angepaßt werden; die separierten Funktionen selbst sind sofort lauffähig. Diese Gruppierung wird überdies von allen agglomerativen Clusterverfahren beim Erreichen der Zielclusteranzahl von drei automatisch erstellt. Des weiteren erlaubt Abbildung 32 das sofortige Erkennen dreier bzgl. des Slicingkriteriums unkohäsiver Gruppen von Daten-Token, von denen allerdings zwei durch das Element N3 miteinander verbunden sind. Bzgl. der Restrukturierung ist sofort ersichtlich, daß das Separieren des Slices(SumSqrN) ohne zusätzliche Modifikation der Funktion durchführbar ist, wohingegen bei den beiden anderen Slices das Verbindungs-Token, das in der Visualisierung die Glue-Metapher sehr gut wiedergibt, berücksichtigt werden muß. Ein Super-Glue-Token befindet sich in solchen Visualisierungen im Zentrum der Visualisierung; mehrere Super-Glue-Token, die aufgrund ihrer besonderen Eigenschaften zueinander alle eine Distanz von Null besitzen, bilden einen Super-Glue-Cluster von Daten-Token.

Nach ähnlichem Muster können ebenfalls viele andere funktionale Kohäsionsmaße wie z.B. *Tightness*, *Overlap*, *Coverage*, *Parallelism* und *Clustering* beschrieben werden (vgl. [OtTh93]). Die oben aufgeführten Vorteile einer distanzmaßbasierten Neuformulierung gelten dort ebenfalls, d.h.

die Analyse der funktionalen Kohäsion kann durch Anwendung des generischen Distanzmaßes und darauf aufbauenden Techniken deutlich optimiert werden

Im folgenden Abschnitt liegt der Fokus auf abstrakter Kohäsion, die vor allen Dingen bei einer Kohäsionsbetrachtung von Entitäten objektorientierter Systeme relevant ist.

7.1.2 Abstrakte Kohäsion

Mit dem Aufkommen der objektorientierten Analyse und des objektorientierten Designs existiert durch die Möglichkeit der Klassendefinition eine zusätzliche Abstraktionsschicht, die es erlaubt, mehrere Funktionen einer Klasse zuzusprechen. Einige Kohäsionsansätze versuchen daher auch, eine Klasse lediglich als Container von Funktionen aufzufassen, bei denen die VRES (s.o.) den einzelnen Methoden entnommen werden und die Ausgabeparameter durch die Attribute gegeben sind: So wird in [OBKM95] z.B. die Betrachtungsweise der funktionalen Kohäsion und deren Maße (vgl. Abschnitt 7.1.1) direkt auf objektorientierte Klassen übertragen; die definierten Maße *Strong Data Cohesion (SDC)* , *Weak Data Cohesion (WDC)* und *Data Adhesiveness* entsprechen dabei jeweils ihren funktionalen Pendants.

Da eine Klasse aber mehr als ein Funktionscontainer sein soll – nämlich die Implementierung eines *abstrakten Datentyps*, d.h. von gekapselten Daten mit erlaubten Operationen darauf – wurde bereits 1987 eine spezielle Kohäsionsabstufung für abstrakte Datentypen hergeleitet [EmWo87], die vom Typus her den sieben Kohäsionsstufen von Myer, Constantine und Stevens entspricht. Mit der von Coad und Yourdon 1991 eingeführten Hierarchie der Kohäsion entsprechend der betrachteten Bestandteile eines Designmodells, die vom feingranularen *Service*, über die *Klasse* bis hin zur *Vererbungsstruktur* reicht ([CoYo94], S. 149ff), sind diese Stufen von Eder, Kappel und Schrefl auf objektorientierte Systeme, und dort besonders intensiv auf Klassen, übertragen worden [EdKaSc93]. Entsprechend dieser Hierarchie wird die abstrakte Kohäsion als zusätzliche Sichtweise auf das System auf höherer Abstraktionsstufe vorgestellt. Es wird explizit dargelegt, daß die Prinzipien der funktionalen Kohäsion nach wie vor gültig sind, und zwar für die einzelnen Methoden (Services in [CoYo94]): Ihre funktionale Kohäsion, die als *Methodenkohäsion* bezeichnet wird, kann wie in Abschnitt 7.1. ermittelt werden und mittels zusätzlicher, nicht automatisch extrahierbarer Daten auf die siebenstufige Kohäsionsabstufung abgebildet werden.

Die zusätzliche Form der Kohäsionsbetrachtung untersucht, inwieweit die Daten und Methoden zusammen einen einzigen abstrakten Datentyp darstellen. Da diese Betrachtung ebenfalls für die Hierarchiestufe „Vererbungsstruktur“ gilt, d.h. auch innerhalb einer Vererbungsstruktur soll jede Klasse bzgl. der selbst definierten und geerbten Methoden und Attribute einen ADT darstellen, wird in [EdKaSc93] nicht weiter zwischen diesen beiden Abstraktionsstufen unterschieden: „*Since the aim for each newly defined subclass is to exhibit a single semantic concept we may use the same classification for inheritance cohesion as it was defined for class cohesion*“ ([ebd.], S. 29).

Die Tatsache, daß ein häufiges Indiz für diese abstrakte Kohäsion auf der Datenzusammengehörigkeit liegt, führte dazu, daß abstrakte Kohäsion häufig auch als *Datenkohäsion* bezeichnet wurde(vgl. [FePf96], S. 313).

Die von [EmWo87] und [EdKaSc93] verwendete Abstufung für abstrakte Kohäsion umfaßt folgende fünf Stufen (vgl. auch [BrDaWü97]):

- *Separierbare abstrakte Kohäsion (separable)*: Die Klasse repräsentiert mehrere, in keinerlei Beziehung stehende Datenabstraktionen. So ist die Kohäsion einer Klasse z.B. separierbar, wenn die Methoden und Attribute derart in zwei Gruppen geteilt werden können, daß jeweils keine Methode Attribute oder Methoden aus der anderen Gruppe verwendet.
- *Mehrfach-facettierte abstrakte Kohäsion (multifaceted)*: Die Klasse repräsentiert mehrere Datenabstraktionen, die durch mindestens eine Methode, die alle Datenabstraktionen verwendet, in Beziehung zueinander stehen. Typische Beispiele sind Klassen, in denen für mehrere Datenabstraktionen gemeinsame, nicht direkt problembezogene Funktionalität

bereitgestellt wird (z.B. Drucken, persistentes Ablegen oder intern optimiertes Verwalten von Objekten).

- *Undelegierte abstrakte Kohäsion (non-delegated)*: Die Klasse besitzt mindestens ein Attribut, das nicht zur gesamten Datenabstraktion gehört, sondern nur zu einem bestimmten Teil davon. Typische Beispiele sind Klassen, bei denen unterschiedliche Rollen der einzelnen Objekte nicht mittels Herausfaktorisieren der Rollencharakteristika in unterschiedliche Klassen realisiert sind, sondern als Vereinigung aller Attribute aller Rollen innerhalb einer Klasse implementiert sind. Je nach Rolle eines Objekts sind dabei nur Teilmengen der Attribute sinnvoll belegt.
- *Verheimlichte abstrakte Kohäsion (concealed)*: Innerhalb der Klasse existiert mindestens eine sinnvolle Datenabstraktion, die aufgrund ihrer eigenen Kohäsion (bzw. die Kohäsion der von ihr genutzten Attribute und Methoden) in eine externe Klasse ausgelagert werden sollte. Typische Beispiele sind Datenabstraktionen, bei denen komplexe Merkmale von Objekten nicht als separate Klasse modelliert sind; dies gilt z.B. für komplexe Merkmale wie „Adresse" oder „Publikationen" einer Klasse „Mitarbeiter".
- *Vorbildliche abstrakte Kohäsion (model)*: Die Klasse repräsentiert einen einzigen, semantisch zusammengehörenden abstrakten Datentyp.

Ein Großteil der Versuche, dieses durch diese Kohäsionsstufen vermittelte Verständnis einer kohäsiven Klasse zu vermessen, beruhen auf Modifikationen des von Chidamber und Kemerer in ihrer 1991 vorgestellten [ChKe91] und 1994 überarbeiteten „*Metrics Suite for Object Oriented Design*" vorgestellten *Lack of Cohesion of Methods (LCOM)* Maßes (vgl. Kapitel 3.2.3 und [ChKe94]).

Das LCOM-Maß läßt sich entsprechend der vorgestellten Kohäsionsdimensionen (vgl. Abschnitt 7.1) wie folgt beschreiben:

- *Art der zu betrachtenden Entität*: Betrachtet wird die Kohäsion von Klassen, d.h. am Ende der Betrachtung soll jede einzelne Klasse einzuschätzen sein, inwieweit sie abstrakte Kohäsion besitzt.
- *Art der Elemente der betrachteten Entität*: Als Elemente der zu vermessenden Entität, deren Zusammengehörigkeit das Kohäsionsmaß begründen, werden Methoden herangezogen. Sowohl in [ChKe94] als auch in vielen Folgearbeiten bleibt sowohl unklar, ob geerbte Methoden mit betrachtet werden, als auch, ob die betrachteten Methoden eine spezielle Sichtbarkeit besitzen müssen.
- *Art der Zusammengehörigkeit der Elemente*: Die Zusammengehörigkeit der Elemente wird aufgrund ihrer gemeinsam benutzten Attribute bestimmt, d.h. zwei Methoden werden als kohäsiv bezeichnet, wenn sie mindestens ein Attribut gemeinsam benutzen.

Aufbauend auf diesem Verständnis werden für eine Klasse C mit den Methoden $\mathbb{M}=\{m_1, m_2, ..., m_n\}$ und den Attributen $\mathbb{A}=\{a_1, a_2, ...,a_k\}$ zwei Mengen definiert:

$\mathbb{P}:= \{(m_i, m_j): m_i, m_j \in \mathbb{M}, 0<i<j\leq n \wedge \neg \exists\ a_i \in \mathbb{A}: (m_i \text{ benutzt } a_i \wedge m_j \text{ benutzt } a_i)$ und
$\mathbb{Q}:= \{(m_i, m_j): m_i, m_j \in \mathbb{M}, 0<i<j\leq n \wedge \exists\ a_i \in \mathbb{A}: (m_i \text{ benutzt } a_i \wedge m_j \text{ benutzt } a_i)$.

Das LCOM-Maß ist dann definiert als $\text{Max}(|\mathbb{P}| - |\mathbb{Q}|, 0)$ (vgl. [ChKe94]) und wird als *inverses Maß* bezeichnet, d.h. eine Klasse ist sehr kohäsiv bei kleinen LCOM-Werten und wenig kohäsiv bei großen LCOM-Werten.
Das LCOM-Maß kann unmittelbar in eine konkrete Instanziierung des generischen Distanzmaßes überführt werden:

- Die Entitäten, deren jeweilige Distanz zueinander bestimmt wird, sind durch die Methoden $m_i \in \mathbb{M}$ gegeben.
- Als Merkmalsträgermenge wird $\mathbb{A}$, d.h. die innerhalb der Klasse definierten Attribute verwendet. Die betrachteten Merkmale der Methoden bestehen aus der Menge lesend oder schreibend benutzter Attribute $a_i \in \mathbb{A}$.

Damit lassen sich die Mengen $\mathbb{P}$ und $\mathbb{Q}$ wie folgt neu formulieren ([SiLöLe99], S. 71f):
$\mathbb{P}:= \{(m_i, m_j): m_i, m_j \in \mathbb{M}, 0<i<j\leq n \wedge \text{dist}(m_i, m_j)=1$
$\mathbb{Q}:= \{(m_i, m_j): m_i, m_j \in \mathbb{M}, 0<i<j\leq n \wedge \text{dist}(m_i, m_j)<1$

Aus dieser Neuformulierung des LCOM-Maßes mittels des generischen Distanzmaßes lassen sich direkt folgende Aussagen ableiten:

- Das verwendete Ähnlichkeitskonzept zwischen Methoden berücksichtigt lediglich direkte Attributsbenutzungen. Die innerhalb des objektorientierten Entwurfs empfohlene Möglichkeit, Attribute nur über explizite Get- und Set-Methoden zu verwenden (vgl. z.B. „*Self Encapsulate Field*" in [Fowl99] und Kapitel 9.3.2), läßt die interne Validierung dieses Maßes für solche Klassen scheitern. Die Möglichkeit, auf Attribute mittels Methoden zuzugreifen, kann darüber hinaus auch indirekt geschehen, d.h. das Attribut wird über mehrere Nachrichtenaufrufe an die das Attribut anfordernde Methode durchgereicht.
- Die Diskretisierung der Distanz auf die beiden Fälle dist $(m_i, m_j)=1$ und dist $(m_i, m_j)<1$ führt zu einem Informationsverlust innerhalb der LCOM-Berechnung: Es wird keine Unterscheidung vorgenommen, ob zwei Methoden ein Attribut gemeinsam verwenden, oder ob sie alle möglichen Attribute der Klasse gemeinsam verwenden.
- Alle Methoden werden gleich gewichtet. Die besondere Rolle von einigen speziellen Methoden wie z.B. Konstruktor und Destruktor, die aus programmiertechnischen Gründen häufig eine Initialisierung bzw. das Löschen vorkommender Attribute übernehmen, führt zur Erhöhung der Gesamtkohäsion (da solche Methoden eine Distanz <1 mit allen anderen, wenigstens ein Attribut verwendenden Methoden besitzen). Sinnvoll ist daher eine Unterteilung in zwei verschiedene Arten von Methoden: „*A method in a class C is **special**, if it is an accessor method, a delegation method, a constructor, or a destructor in a class C. A method is called **normal** if it is not special*" ([ChKw98], S. 160).
- Die verwendete Form der Bewertung von Paarwerten hängt von der Anzahl der Methoden ab (für n Methoden existieren $\binom{n}{2}$ Methodenpaare).

  Als Beispiel kann folgende, in Abbildung 33 abgebildete Struktur von Methoden und jeweils verwendeten Attributen in Form von Venn-Diagrammen verwendet werden (vgl. Kapitel 6.1), bei denen die benutzten Attribute die Mengenzugehörigkeit der Methoden angeben (vgl. auch [HiMo95]):

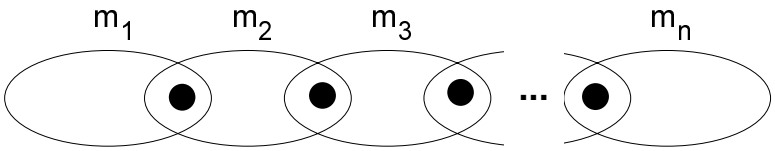


Abbildung 33: Venn-Diagramm für gemeinsam benutzte Attribute von Methoden

Klassen, deren Methoden diese Struktur aufweisen, besitzen für weniger als 5 Methoden einen LCOM-Wert von 0. Für 5, 6, 7 oder 8 Methoden ergibt sich ein LCOM-Wert von 2, 5, 9 bzw.

14. Die durch das Maß widergespiegelte Kohäsionsverschlechterung besitzt keine Entsprechung im empirischen Relationensystem (vgl. [ebd.]).

- Die Bewertungsfunktion ψ, die alle (skalierten) Einzeldistanzen durch die Differenzbildung der Kardinalitäten zweier Mengen auf einen Gesamtkohäsionswert abbildet, führt – wie bereits bei den Maßen zur Bestimmung der funktionalen Kohäsion – zu einem Informationsverlust der Ergebnisse: Diese Bewertung läßt nur noch wenig Rückschlüsse auf die einzelnen Mengen, deren Elemente und Kardinalitäten zu.
- Ein schlechter Kohäsionswert läßt kaum eine Analyse bzgl. möglicher konstruktiver Restrukturierungen zur Erhöhung der Kohäsion zu.

Diese, durch die Neudefinition mittels des generischen Distanzmaßes jetzt offensichtlichen Defizite des Original-LCOM-Maßes führten in der Vergangenheit zu vielen einzelnen LCOM-Modifikationen. Einige sollen kurz vorgestellt werden und erneut mittels des generischen Distanzmaßes reformuliert werden, was einerseits wiederum das einfache Erkennen noch vorhandener Schwächen und andererseits die einfache und einheitliche Gegenüberstellung mit dem Originalmaß ermöglicht.

- Erweiterung um direkte Methodenbenutzungen ([HiMo95]): Um die Ermittlung abstrakter Kohäsion mittels direkter Attributsbenutzungen um indirekte Attributsbenutzungen mittels expliziter Zugriffsmethoden zu erweitern, muß lediglich die Merkmalsträgermenge $\mathbb{A}$ modifiziert und um die innerhalb der Klasse definierten Methoden erweitert werden. Die betrachteten Merkmale der Methoden bestehen aus der Menge lesend oder schreibend benutzter Attribute $a_i \in \mathbb{A}$ und der benutzten Methoden $m_i \in \mathbb{M}$.
- Erweiterung um (in-)direkte Methodenbenutzungen ([BiKa95]): Als weitere Erweiterung des LCOM-Maßes berücksichtigen Biemann und Kang ebenfalls die Möglichkeit, über mehrere Indirektionen von Methodenaufrufen Attribute zu verwenden. Auf Basis der Festlegung, daß zwei Methoden *direkt kohäsiv* sind, wenn sie wenigstens ein Attribut gemeinsam verwenden, werden zwei Methoden als *indirekt kohäsiv* bezeichnet, wenn sie beide in einer transitiven Hülle der direkt kohäsiven Methoden liegen. So gilt z.B. für die Methoden aus Abbildung 33, daß jeweils zwei Methoden m_i und m_{i+1} direkt kohäsiv sind; zwei Methoden m_i und m_j mit $|i-j|>1$ sind dagegen jeweils indirekt kohäsiv. Darauf aufbauend wird das Maß *Loose Class Cohesion* (LCC) wie folgt definiert (die Menge der direkt oder indirekt kohäsiven Methoden im Nenner wird häufig als die *common attribute usage* (*cau*)-Menge bezeichnet):

$$LCC(C) := \frac{|\{(m_i, m_j) : m_i, m_j \in M \wedge 0 < i < j \leq \ n \wedge m_i \, und \, m_j \, sind \, direkt \, oder \, indirekt \, kohäsiv\}|}{\sum_{k=1}^{n} k}$$

Dabei sind die m_i die n Methoden der Klasse C.

Für eine Neuformulierung dieses Maßes mittels des generischen Distanzmaßes muß lediglich die Merkmalsträgermenge dahingehend modifiziert werden, daß jede Methode nicht nur die direkt benutzten Attribute als Merkmale besitzt, sondern darüber hinaus diejenigen Attribute enthält, die transitiv mittels benutzter Methoden indirekt verfügbar sind. So besitzt jede Methode aus Abbildung 33 alle Attribute als Merkmal.

LCC selbst wird nicht auf alle Methoden angewendet, sondern lediglich auf öffentlich sichtbare, die weder Konstruktor noch Destruktor sind. Innerhalb einer Neuformulierung kann dies durch eine Modifikation der Menge $\mathbb{M}$ erreicht werden.

- Trennung von Kohäsionsgüte und Anzahl betrachteter Methoden ([LiHe93], optimiert in [BiKa95]): Um das Kohäsionmaß nicht direkt von der Anzahl betrachteter Methoden abhängig zu machen – was bei der Betrachtung von Methodenpaaren mit bestimmten Eigenschaften i.d.R. der Fall ist – und um gleichzeitig durch den Meßwert eine bessere

Vorstellung davon zu bekommen, in wieviele Teilklassen die Klasse bei einem schlechten Kohäsionswert aufzuteilen ist, haben Li und Henry das LCOM-Maß graphtheoretisch modifiziert: Für eine Klasse C wird ein gerichteter Graph G_C ($\mathcal{V}$, $\mathcal{E}$) eingeführt, wobei
$\mathcal{V}$= Menge aller Methoden in C, d.h. $\mathcal{V}=\mathbb{M}$, und
$\mathcal{E}=\{<m_i, m_j> \in \mathcal{V}\times\mathcal{V} \mid \exists$ Attribut $a\in\mathbb{A}$: (m_i benutzt a) $\wedge$ (m_j benutzt a)$\}$.
„*LCOM (C) is then defined as the number of connected components of* G_C *(*$1 \leq$ *LCOM(C)* $\leq |\mathbb{M}_C|$*)* " ([HiMo95], S. 9), d.h. es wird die Anzahl *streng verbundener Teilgraphen* bestimmt (vgl. z.B. [TrMa75], S. 480f). Dieses LCOM-Maß ergibt für alle Klassen mit einer Struktur, die derjenigen in Abbildung 33 entspricht, einen Kohäsionswert von 1.
Wird die Merkmalsträgermenge wie im letzten Punkt um diejenigen Attribute erweitert, die transitiv mittels benutzter Methoden indirekt verfügbar sind, so kann die Zugehörigkeit zweier Methoden zu unterschiedlichen Teilgraphen dadurch bestimmt werden, daß ihre Distanz zueinander eins ist. Die Anzahl verbundener Teilgraphen ist bestimmt durch die Anzahl derjenigen Mengen von Methoden, die paarweise zueinander jeweils die Distanz eins besitzen.
Die Erweiterung des Graphen um Kanten, die eine direkte Benutztbeziehung zwischen Methoden (ohne Berücksichtigung eventuell gemeinsam verwendeter Attribute) widerspiegeln (vgl. [HiMo95]), kann wiederum durch eine Anpassung der Merkmalsträgermenge für die distanzmaßbasierte Neuformulierung nachgezogen werden.

- Prozentuale Kohäsion ([Hend96]): Für eine einfachere Analyse der Kohäsionsmeßwerte schlägt Hendersson-Sellers vor, „[...] *that a better LCOM measure should have values on a percentage range*" ([Hend96], S. 147). Die Einteilung der Kohäsion soll dabei den prozentualen Anteil einer idealen „perfekten Kohäsion" widerspiegeln, die genau dann gegeben ist, wenn alle Methoden einer Klasse alle Attribute derselben Klasse verwenden. Eine Klasse besitzt 0% dieser perfekten Kohäsion, wenn jede Methode ihr eigenes Attribut verwendet.
Auf Grundlage einer Funktion $\mu(a_i)$, die für ein Attribut $a_i \in \mathbb{A}$ die Anzahl es benutzender Methoden angibt, ist auf Grundlage der oben definierten Mengen $\mathbb{M}=\{m_1, m_2, ..., m_n\}$ und $\mathbb{A}=\{a_1, a_2, ...,a_k\}$ das Maß LCOM* definiert als ([ebd.], S. 147):

$$LCOM^*(C) := \frac{\left(\frac{1}{k} * \sum_{p=1}^{k} \mu(a_p)\right) - n}{1 - n}$$

Aufgrund des inversen Charakters von LCOM, das schließlich das Fehlen (lack) von Kohäsion bestimmen soll, gibt dieser Meßwert den prozentualen Anteil des Fehlens der perfekten Kohäsion an, d.h. ein LCOM*-Wert von 0 steht für perfekt kohäsive Klassen (0% Fehlen von der perfekten Kohäsion), wohin gegen ein LCOM*-Wert von 1 für eine vollständig unkohäsive Klasse steht.
Bei der Nachbildung dieses Maßes mittels des generischen Distanzmaßes fällt auf, daß diese LCOM-Modifikation nicht von den Methoden (und deren Ähnlichkeiten über gemeinsam verwendete Attribute) ausgeht, sondern von den Attributen direkt, d.h. die Elemente, deren jeweilige Distanz zueinander bestimmt werden muß, sind durch die Attribute $a_i \in \mathbb{A}$ gegeben.
Als Merkmalsträgermenge wird $\mathbb{M}$, d.h. die innerhalb der Klasse definierten Methoden verwendet. Die betrachteten Merkmale der Attribute bestehen aus der Menge sie lesend oder schreibend benutzender Methoden. Des weiteren fällt auf, daß nicht die Distanzen der Attribute zueinander relevant sind – schließlich wird nur die Anzahl der ein Attribut verwendenden Methoden gezählt, die für zwei unterschiedliche Attribute durchaus disjunkt sein dürfen – sondern die Distanzen zu einem imaginären, perfekt kohäsiven Attribut a_{cpl}, das von allen Methoden benutzt wird. Damit gilt:

$$\frac{1}{k}\sum_{p=1}^{k}\mu(a_p) = \frac{1}{k}\sum_{p=1}^{k} dist(a_p, a_{cpl}) * k = \sum_{p=1}^{k} dist(a_p, a_{cpl}) \ .$$

Auch wenn viele der dem Original-LCOM-Maß zugeschriebenen Schwachpunkte in einigen Modifikationen behoben sind, so sind die beiden wichtigen Punkte des Informationsverlustes durch die Bewertungsfunktion ψ und die fehlende Möglichkeit einer konstruktiven Analyse in allen Versionen nach wie vor vorhanden. In Kapitel 8 wird daher ein Verfahren vorgestellt, das viele Aspekte der in den verschiedenen vorgestellten Kohäsionsmaßen – wie z.B. die für die Reformulierung notwendigen instanziierten Distanzmaße – wiederverwendet, allerdings auf das Zusammenfassen zu einem Einzelwert in Form der Bewertung verzichtet. Die dadurch gegebene Möglichkeit der aussagekräftigen Visualisierung und die Ableitung konstruktiver Restrukturierungsschritte wird dort ebenfalls anhand einiger Beispiele demonstriert.

Die Wiederverwendung bekannter Kohäsionskonzepte beschränkt sich daher im wesentlichen auf die Art der betrachteten Elemente und die Art der Zusammengehörigkeit der Elemente. Um für die letztere Dimension eventuell zusätzliche Ansätze zu identifizieren, die für eine Kohäsionsbestimmung mittels des generischen Distanzmaßes relevant sein können, wird im folgenden Abschnitt die Kopplung separat betrachtet, da sie sich intensiv mit verschiedenen Arten der Abhängigkeiten zwischen Elementen beschäftigt und damit Kohäsion begründen kann.

7.2 Kopplung

Während Kohäsion den Grad der Zusammengehörigkeit von Elementen betrachtet, beschäftigt sich die Kopplung mit verschiedenen Arten der Abhängigkeit von Elementen zueinander. Dem allgemeinen Begriff der Kopplung kann innerhalb der Softwaretechnik folgende Definition zugrunde gelegt werden:

> Kopplung betrachtet die Art und den Grad der Abhängigkeit zwischen mehreren betrachteten Elementen.

Definition 34: Kopplung (vgl. [IEEE90])

Unter der Prämisse, daß mehrere Elemente mit einer hohen Abhängigkeit zueinander, d.h. einer hohen gegenseitigen Kopplung, auch eine hohe Zusammengehörigkeit, d.h. eine hohe Kohäsion besitzen, kann Kopplung als konkreter Parameter für die *Art der Zusammengehörigkeit* bei der Bestimmung von Kohäsion (vgl. Abschnitt 7.1) verwendet werden: Kopplung kann Kohäsion begründen!
Nachdem Kohäsion im letzten Abschnitt durch die Bewertung der durch Anwendung spezifisch instanziierter Distanzmaße ermittelten Distanzwerte nachgebildet wurde, geht es jetzt darum, die verschiedenen Betrachtungsweisen von Kopplung vorzustellen, um darauf aufbauend dann zusätzliche Kohäsionsformen herleiten zu können.
Die verschiedenen Kopplung messenden Maße brauchen im folgenden nicht weiter behandelt zu werden, da für die Zwecke dieses Abschnitts, nämlich der Identifikation möglichst vielseitiger Aspekte für die Anwendung der distanzmaßbasierten Kohäsionsbestimmung, ihr Charakteristikum, einzelnen Knoten Werten zuzuweisen (vgl. Kapitel 5.1.2), irrelevant ist. Statt dessen werden die dahinter stehenden *Kopplungskonzepte* vorgestellt, d.h. Arten von Abhängigkeiten, die später für die Kohäsionsbetrachtung sinnvoll sein können. Damit wird es erstmals möglich sein, die ansonsten getrennt behandelten Konzepte Kopplung und Kohäsion mittels des Distanzkonzepts einheitlich zu betrachten und damit das Kopplungs-Kohäsions-Problem lösen zu können, indem Kohäsion als Bewertung der auf Grundlage von Kopplung ermittelten Distanzen aufgefaßt wird. Auf Grundlage dieser Abstraktion kann in Abschnitt 7.3 ein generischer Prozeß zur distanzmaßbasierten (und Kopplung berücksichtigenden)

Kohäsionsbetrachtung vorgestellt werden. Im Folgenden soll vorher ein Überblick über verschiedene Arten von Kopplung gegeben werden, die entsprechend einer Hierarchie objektorientierter Systeme in Interaktionskopplung (via Funktionen oder Methoden), Klassenkopplung und Vererbungskopplung verfeinert werden kann.

7.2.1 Interaktionskopplung

Das Konzept der Kopplung als Qualitätsmerkmal wurde erstmals von Stevens, Myers und Constantine explizit behandelt [StMyCo74]. Ähnlich des Kohäsionsspektrums identifizierten sie verschiedene Niveaus und ordneten sie ordinal bzgl. des Kopplungsgrads. Die sechs dort identifizierten Kopplungsniveaus sind (von der stärksten Kopplung zur schwächsten):

- *Inhaltliche Kopplung (content)*: Hierbei werden Inhalte einer Entität direkt von einer anderen referenziert. Typische Beispiele sind Sprungmarken oder der Zugriff auf Daten fremder Entitäten mittels absoluter Adressangaben.
- *Allgemeine Kopplung (common)*: Hier teilen sich mehrere Entitäten allgemein bekannte, global definierte Daten, d.h. sie schreiben und lesen diese. Beispiele sind global definierte, unstrukturierte Datensammlungen in C.
- *Externe Kopplung (extern)*: Wie bei der allgemeinen Kopplung teilen sich mehrere Entitäten gemeinsame Daten, allerdings sind diese bei der externen Kopplung strukturiert (z.B. als Array) und explizit für eine externe Benutzung deklariert (z.B. öffentlich sichtbare (`public`) Attribute innerhalb einer C++- oder JAVA-Klasse)
- *Steuernde Kopplung (control)*: Hier kommuniziert eine Entität mit einer anderen und nimmt damit Einfluß auf deren Kontrollfluß. Beispiele sind codierte Rückgabeparameter (z.B. bei typischen I/O-Operationen), auf die entsprechend der Codierung reagiert werden muß.
- *Strukturelle Kopplung (stamp)*: Hierbei verwenden mehrere Entitäten dieselbe nicht-globale Datenstruktur, die sich aus verschiedenen Einzelkomponenten zusammensetzt. Beispiele sind übergebene records/structs, die sich aus komplexen Einzeldaten zusammensetzen (z.B. ein Adressen-Record).
- *Datenbasierte Kopplung (data)*: Hierbei verwenden mehrere Entitäten dasselbe nicht-globale Datenelement oder eine Menge homogener Datenelemente. Die einzelnen Datenelemente besitzen keine darüber hinausgehende Struktur. Beispiele sind einfache Parameter bei Funktionsaufrufen (z.B. Integer-Werte).

Das durch diese sechs Stufen aufgespannte Kopplungsspektrum kann wie in Abbildung 34 dargestellt werden:

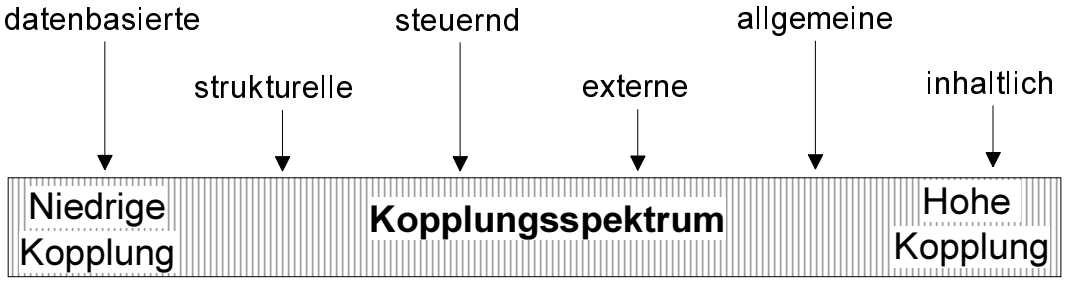


Abbildung 34: Kopplungsspektrum (vgl. [Pres97], S. 359)

Wie das Kohäsionsspektrum innerhalb des objektorientierten Vorgehens und dessen Modellen weiter verwendet und lediglich um zusätzliche, die neuen Abstraktionsebenen betreffenden Spektren angereichert wurde, so kann auch das oben aufgeführte Kopplungsspektrum auf objektorientierte Systeme übertragen werden. Die dafür betrachteten Elemente sind dabei durch die Methoden gegeben, d.h. Methoden können theoretisch alle Grade zwischen inhaltlicher und

datenbasierter Kopplung einnehmen; moderne Programmiersprachen unterstützen allerdings häufig nicht mehr direkt alle Möglichkeiten. So kann z.B. die inhaltliche Kopplung in C++ nur noch künstlich über Funktionspointer und deren Modifizierung erreicht werden.
Die von [EdKaSc93] vorgeschlagene Klassifikation von Kopplung innerhalb objektorientierter Systeme führt neben dieser *Interaktionskopplung* (*Interaction Coupling*), für die das von Stevens, Myer und Constantine aufgezeigte Kopplungsspektrum nach wie vor gültig ist, zwei zusätzliche Kopplungsarten ein (vgl. [BrDaWü99]):

- *Komponentenkopplung (Component Coupling)*, bei der die eine Entität Teil der anderen ist und damit ihre Elemente zur Verfügung stellt, und
- *Vererbungskopplung (Inheritance Coupling)*, bei der zwei Klassen in einer Vererbungsbeziehung stehen.

Für jede Art wird ein separates Kopplungsspektrum erstellt, das im folgenden jeweils kurz vorgestellt werden soll.

7.2.2 Komponentenkopplung.

Unter der Prämisse, daß innerhalb objektorientierter Systeme die Komponenten durch die Klassen gegeben sind und die Methoden als Teile davon angesehen werden, stellt die Komponentenkopplung eine Kopplungsbetrachtung auf dem, in Bezug zum Niveau der Interaktionskopplung, nächst höheren Abstraktionsniveau dar. Da Klassen selbst keine Abhängigkeiten aufweisen (Ausnahme: Vererbung, s. nächster Abschnitt), sondern durch Abhängigkeiten der Elemente gegeben sind, ist Komponentenkopplung nur auf Grundlage einer Interaktionskopplung möglich: sie basiert auf ihr. Die Komponentenkopplung betrachtet lediglich, wie explizit das Verwenden fremder Klassen in Form benötigter Methoden ist. Drei verschiedene Kopplungsarten, geordnet von der stärksten Kopplung hin zur schwächsten Kopplung, werden daher innerhalb der Komponentenkopplung unterschieden (im weiteren wird dabei der Komponentenbegriff jeweils durch den spezialisierten Klassenbegriff ersetzt):

- *Versteckte Klassenkopplung (hidden)*: Eine Klasse C_1, innerhalb der ein Element existiert, das ein Element aus einer Klasse C_2 verwendet, ist versteckt mit dieser Klasse C_2 gekoppelt, wenn C_2 weder im Deklarationsteil von C_1, noch innerhalb der Implementierung eines Elements von C_1 explizit erscheint. Typische Beispiele einer solchen Kopplung sind Methodenaufrufe, bei denen die Rückgabewerte direkt als Eingabewerte für andere Methodenaufrufe verwendet werden.
- *Verteilte Klassenkopplung (scattered)*: Eine Klasse C_1 ist verteilt mit einer Klasse C_2 gekoppelt, wenn C_2 nicht im Deklarationsteil von C_1 erscheint, es aber mindestens eine Implementierung eines Elements von C_1 existiert, in der C_2 explizit erscheint. Für das Verstehen der Klassenkopplung ist es daher notwendig, alle Implementierungen der Elemente zu überprüfen. Typische Beispiele einer solchen Kopplung sind Helferklassen, die Funktionalität anbieten, die nur von einem kleinen Teil der Methoden der aufrufenden Klasse benötigt werden.
- *Spezifizierte Klassenkopplung (specified)*: Eine Klasse C_1 ist spezifiziert mit einer Klasse C_2 gekoppelt, wenn C_2 explizit im Deklarationsteil von C_1, z.B. in Form eines privaten Attributs, erscheint.

Da das automatische Erkennen des Grades der Klassenkopplung sehr schwierig ist, beschränken sich die Kopplung zwischen Klassen betrachtenden Maße i.d.R. auf die numerische Angabe, zu wieviel anderen Klassen eine Klasse eine versteckte, verteilte oder spezifizierte Kopplung besitzt.

Typische Maße für diese Betrachtungsweise von Komponentenkopplung sind z.B. das Maß *coupling between objects (CBO)* [ChKe94], das für eine Klasse die Anzahl anderer, mit ihr bzgl. einer Benutzung gekoppelter Klassen zählt (sowohl Methoden- als auch Attributsbenutzung), das Maß *Class-Attribute-Interaction-Count CAIC*[21] [BrDeMe97], das für eine Klasse alle anderen Klassen zählt, die als Attribut (lokal in Methoden oder im Deklarationsteil der Klasse) verwendet werden, oder das Maß *Actual Method-Method Interaction (AMMI)* [BrDeMe96], das alle Paare von Benutzungsbeziehungen zwischen zwei Klassen zählt.

7.2.3 Vererbungskopplung

Die innerhalb der Kopplungshierarchie objektorientierter Systeme höchste Stufe, die in [EdKaSc93] als relevant für eine Kopplungsbetrachtung aufgeführt wird, ist die Vererbungskopplung. Wie bei der Komponentenkopplung werden hierbei Klassen betrachtet. Die Abhängigkeit ist allerdings nicht durch direkte oder indirekte Benutzungen gegeben: „*Two classes are inheritance coupled if one class is a direct or indirect subclass of the other* " ([ebd.], S. 16). Die Abhängigkeit innerhalb einer Vererbungsstruktur ist dadurch gegeben, daß Unterklassen sich wie Oberklassen (bzw. die jeweiligen Objekte) verhalten. Das Verhalten ergibt sich dabei aus der Menge verfügbarer Operationen und ihnen zugrundeliegenden Daten. Diese „ist ein"-Relation schließt ein, daß Unterklassen abhängig von Änderungen innerhalb der Superklasse sind. Für den Grad dieser Art von Abhängigkeit werden fünf ordinal geordnete Stufen angegeben (von der stärksten Kopplung zur schwächsten Kopplung):

- *Signaturmodifizierende Vererbungskopplung (Signature Modification)*: Eine Unterklasse C_2 ist signatur-modifizierend mit einer ihrer Oberklassen C_1 gekoppelt, wenn C_2 das geerbte Verhalten grundlegend modifiziert. Typische Beispiele einer signatur-modifizierenden Vererbungskopplung sind das Ändern einer Methodensignatur oder das vollständige Löschen einer geerbten Methode (in C++ z.B. durch den Versuch der Überladung (overloading) von geerbten Methoden, wodurch letztere in der Unterklasse nicht mehr zur Verfügung stehen, vgl. [SiBe00]).
- *Implementierungmodifizierende Vererbungskopplung (Implementation Modification)*: Eine Unterklasse C_2 ist implementierungmodifizierend mit einer ihrer Oberklassen C_1 gekoppelt, wenn C_2 die Semantik geerbter Methoden grundlegend neu implementiert. Typische Beispiele einer implementierungmodifizierenden Vererbungskopplung sind das Löschen von Implementierungen unter Beibehaltung der Signatur oder die Neu-Implementierung zu allgemein bezeichneter und damit semantisch unklarer Methoden (z.B. ‚`do`').
- *Signaturverfeinernde Vererbungskopplung (Signature Refinement)*: Eine Unterklasse C_2 ist signaturverfeinernd mit einer ihrer Oberklassen C_1 gekoppelt, wenn C_2 die Signatur einer geerbten Methode nach vorher definierten Regeln modifiziert. Beispiele solcher definierten Regeln sind die *Kovarianz-Regel* und die *Kontravarianz-Regel* für Unterklassen: Die Kovarianz-Regel erlaubt eine Signaturverfeinerung durch ein Ersetzen der Parameter und des Rückgabewertes durch Unterklassen. Die Kontravarianz-Regel erlaubt die Signaturverfeinerung durch ein Ersetzen der Parameter durch Superklassen und ein Ersetzen des Rückgabewertes durch Unterklassen (vgl. z.B. [Meye97], S. 621ff).

[21] In der Originalarbeit von Briand, Devanbu und Melo [BrDeMe97] sind Maße für die konkrete Programmiersprache C++ angegeben. Diese berücksichtigen explizit die Möglichkeit, mittels des friend-Operators neue Sichtbarkeiten zu erzeugen. Um diese für einige Kopplungsmaße ausschließlich zu berücksichtigen, werden nur diejenigen Klassen berücksichtigt, zu denen eine friend-Relation existiert. Das Maß heißt daher dort vollständig *Inverse-Friends-Class-Attribute-Interaction-Count (IFCAIC)*. Kopplungen zu Klassen, die weder friend sind noch in einer Vererbungsbeziehung zur vermessenen Klasse stehen, werden mittels des Other-Class-Attribute-Interaction-Count(OCAIC) bezeichnet. Das Prinzip der Class-Attribute-Interaction (s.o.) bleibt jedoch bei allen Maßen fix.

- *Implementierungverfeinernde Vererbungskopplung (Implementation Refinement)*: Eine Unterklasse C_2 ist implementierungverfeinernd mit einer ihrer Oberklassen C_1 gekoppelt, wenn C_2 mindestens eine geerbte Methode neu implementiert, wobei allerdings die Semantik, die z.B. mittels Vor- und Nachbedingungen spezifiziert sein kann, invariant bleibt. Typische Beispiele einer implementierungverfeinernden Vererbungskopplung sind bzgl. der Performance optimierte Spezialisierungen in Methoden der Unterklassen, die spezielle Eigenschaften der Unterklasse ausnutzen (z.B. verschiedene Sortieralgorithmen innerhalb einer Vererbungsstruktur verschiedener Container-Klassen).
- *Erweiternde Vererbungskopplung (extension)*: Eine Unterklasse C_2 ist erweiternd mit einer ihrer Oberklassen C_1 gekoppelt, wenn C_2 das geerbte Verhalten weder verfeinert, noch modifiziert, dafür aber eventuell zusätzliches Verhalten hinzufügt. Typische Beispiele einer erweiternden Vererbungskopplung sind Vererbungshierarchien, in denen die Superklasse eine wirkliche semantische Verallgemeinerung ihrer Unterklassen darstellt, d.h. in Unterklassen nur noch das in der Verallgemeinerung für die Spezialisierung fehlende Verhalten implementiert wird.

Wie bereits bei der Klassenkopplung so sind auch hier die Kopplungsgrade nicht vollständig automatisch bestimmbar: So läßt sich z.B. ohne explizite Angabe einer Semantik in Form von z.B. Vor- und Nachbedingungen für die implementierungmodifizierende Vererbungskopplung nicht bestimmen, ob die Neuimplementierung einer Methode semantikerhaltend ist oder nicht. Da sich im Gegensatz dazu leicht erkennen läßt, ob Signaturen hinzugekommen sind oder Methoden neu implementiert wurden (ohne Angabe bzgl. der Semantikerhaltung), beschränken sich Maße für die Vererbungskopplung auf das Betrachten von Sichtbarkeiten und neu implementierten Methoden in einer Klasse. Typische Beispiele sind das Maß *Number of Methods overridden by a Subclass* [LoKi94], das für eine Oberklasse bestimmt, wieviele ihrer Methoden in Unterklassen überschrieben, d.h. neu implementiert werden, das Maß *Number of Methods added by a Subclass* [ebd.], das für eine Unterklasse bestimmt, wieviele Methoden sie der geerbten Menge hinzufügt, oder das Maß *Specialization Index* [ebd.], das für jede Klasse berechnet wird als

$$\frac{\textit{Anzahl überdeckter Methoden der Klasse} * \textit{Vererbungstiefe der Klasse}}{\textit{Gesamtanzahl von Methoden der Klasse}}$$ und für alle Klassen kleiner 15% sein sollte ([ebd.], S. 73). Der Wert selbst wird dabei nicht weiter begründet.

Alle diese Kopplungsarten, d.h. die Interaktionskopplung, die Komponentenkopplung und die Vererbungskopplung, können Kohäsion begründen. Im folgenden soll ein entsprechender Prozeß zur distanzmaßbasierten Kohäsionsbestimmung vorgestellt werden, der in Abschnitt 7.3.1 den speziellen Fall der hier vorgestellten Kopplungen berücksichtigt.

7.3 Prozeß zur distanzmaßbasierten Kohäsionsbestimmung

Allen Betrachtungen von Kohäsion ist gemeinsam, daß sie sich als übergeordnetes Qualitätsmerkmal einer Entität durch eine Analyse der Zusammengehörigkeit der darin enthaltenen Elemente bestimmen läßt. Die folglich für eine Kohäsionsbestimmung notwendige Dimension der „Art der Zusammengehörigkeit der Elemente“ (vgl. Abschnitt 7.1) entspricht üblicherweise den verschiedenen Arten von Kopplung (vgl. Abschnitt 7.2). Da Kohäsion allerdings nicht auf kopplungbasierte Zusammengehörigkeit beschränkt ist, sind darüber hinaus weitere, nicht durch Kopplung begründete Zusammengehörigkeiten der Elemente für eine Kohäsionsbestimmung verwendbar. Für einen Prozeß zur distanzmaßbasierten Kohäsionsbestimmung wird daher die Nachbildung der kopplungbasierten Kohäsion als Spezialfall einer allgemeinen, distanzmaßbasierten Kohäsionsbestimmung aufgefaßt. Im folgenden Abschnitt wird diese Spezialisierung konkret vorgestellt, da sie die Nachbildung der bekannten Kohäsionsmaße erlaubt und aufgrund der Gewichtigkeit des Qualitätsmerkmals

Kopplung den Schwerpunkt der Kohäsionsbetrachtung darstellt. Um aber auch darüber hinaus gehende Aspekte beleuchten zu können, wird im letzten Abschnitt ein allgemeiner Prozeß beschrieben, der neben Kopplung auch weitere Arten der Zusammengehörigkeit von Elementen vorsieht.

7.3.1 Prozeß zur kopplungbasierten Kohäsionsbestimmung mittels des generischen Distanzmaßes

Wird das Konzept Kopplung der Zusammengehörigkeit der Elemente zugrunde gelegt, so kann die kopplungbasierte Kohäsionsbestimmung wie in Abbildung 35 dargestellt modelliert werden (vgl. auch Kapitel 5.1.3):

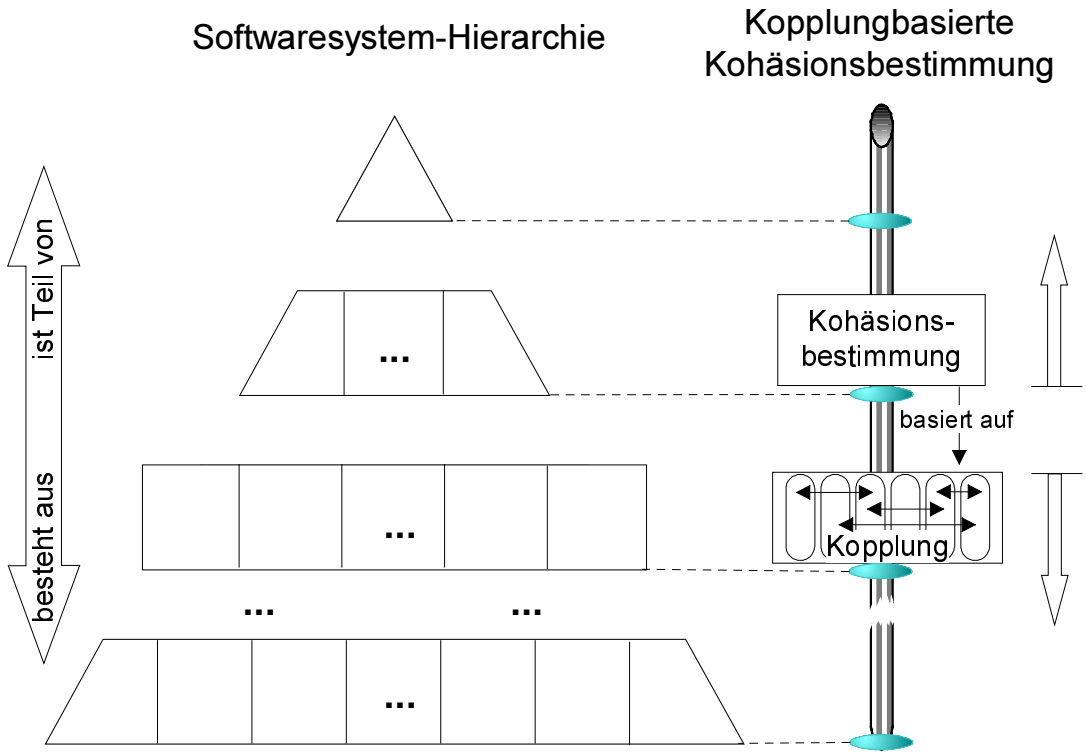


Abbildung 35: Modell zur kopplungbasierten Kohäsionsbestimmung

Die Abbildung verdeutlicht dabei folgende Charakteristika der kopplungbasierten Kohäsionsbestimmung:

1. Die Kopplung der Elemente der betrachteten Entität ist Grundlage für die Kohäsionsbestimmung. Eine kohäsive Entität ist durch stark gekoppelte Elemente gekennzeichnet.
2. Verschiedene Kopplungsarten führen zu unterschiedlichen Kohäsionsarten.
3. Die für die Kohäsionsbestimmung einer Entität relevante Kopplung betrifft Elemente der Entität. Zwischen den Entitäten und den Elementen können sich auch mehrere Hierarchiestufen befinden. Die Hierarchiestufe der bzgl. Kohäsion zu betrachtenden Entität ist frei wählbar, solange es noch eine Verfeinerung für die Betrachtung der Elemente gibt, d.h. Kohäsion ist ein allgemeines Konzept.
4. Typische Kohäsionsmaße (vgl. Abschnitt 7.1) verstehen sich als Bewertung (z.B. durch Summation) der Kopplungen der betrachteten Elemente.

Für ein allgemeines Vorgehen zur Nachbildung einer kopplungbasierten Kohäsionsbestimmung mittels des generischen Distanzmaßes ergeben sich für die jeweiligen Charakteristika folgende Forderungen:

1. Das generische Distanzmaß setzt bei der Umsetzung des jeweiligen Kopplungsverständnisses an und fordert die Definition der Merkmalsträgermenge für die spätere Berechnung der Distanzen zwischen den Elementen. Die Merkmalsträgermenge muß dabei derart gewählt werden, daß stark gekoppelte Elemente viele gemeinsame Merkmale, d.h. eine hohe Ähnlichkeit und damit eine geringe Distanz zueinander besitzen. Dieses Konzept wurde für einige Kohäsionsmaße bereits in Abschnitt 7.1 angewendet. Die vorgestellten Kopplungsarten aus Abschnitt 7.2. ermöglichen darüber hinaus weitere Kohäsionbetrachtungen.
2. Jede Betrachtung von Kohäsion verlangt implizit nach der sie begründenden Kopplungsart, d.h. nach der Art und Weise, in der die Elemente als gekoppelt betrachtet werden (vgl. Abschnitt 7.2).
3. Für die Kohäsionsbestimmung einer Entität wird das generische Distanzmaß auf deren Elemente (oder einer weiteren Verfeinerung davon) angewendet.
4. Neben den Vorteilen einer einheitlichen Nomenklatur für die verschiedenen Kohäsions- und Kopplungsbetrachtungen benötigt die kopplungbasierte Kohäsionsbestimmung mittels des generischen Distanzmaßes keine numerische Bewertungsfunktion. Statt dessen ermöglicht das Distanzkonzept die aussagekräftige Visualisierung und die Anwendung der Clusteranalyse für eine qualitative Identifikation kohäsiver Elementgruppen innerhalb der betrachteten Entität.

Auf Grundlage dieser Forderungen wird ein Prozeß zur kopplungbasierten Kohäsionsbestimmung mittels des generischen Distanzmaßes vorgestellt. Dabei wird nicht mehr zwischen dem Einsatz existierender Kohäsionsbetrachtungen und dem Erstellen neuer Kohäsionsarten mittels spezieller Kopplungsarten unterschieden, da beiden das Modell der kopplungbasierten Kohäsionsbestimmung (vgl. Abbildung 35) zugrunde gelegt werden kann.
Der Prozeß mit seinen einzelnen Arbeitsschritten ist mittels eines Aktivitätendiagramms in Abbildung 36 beschrieben.

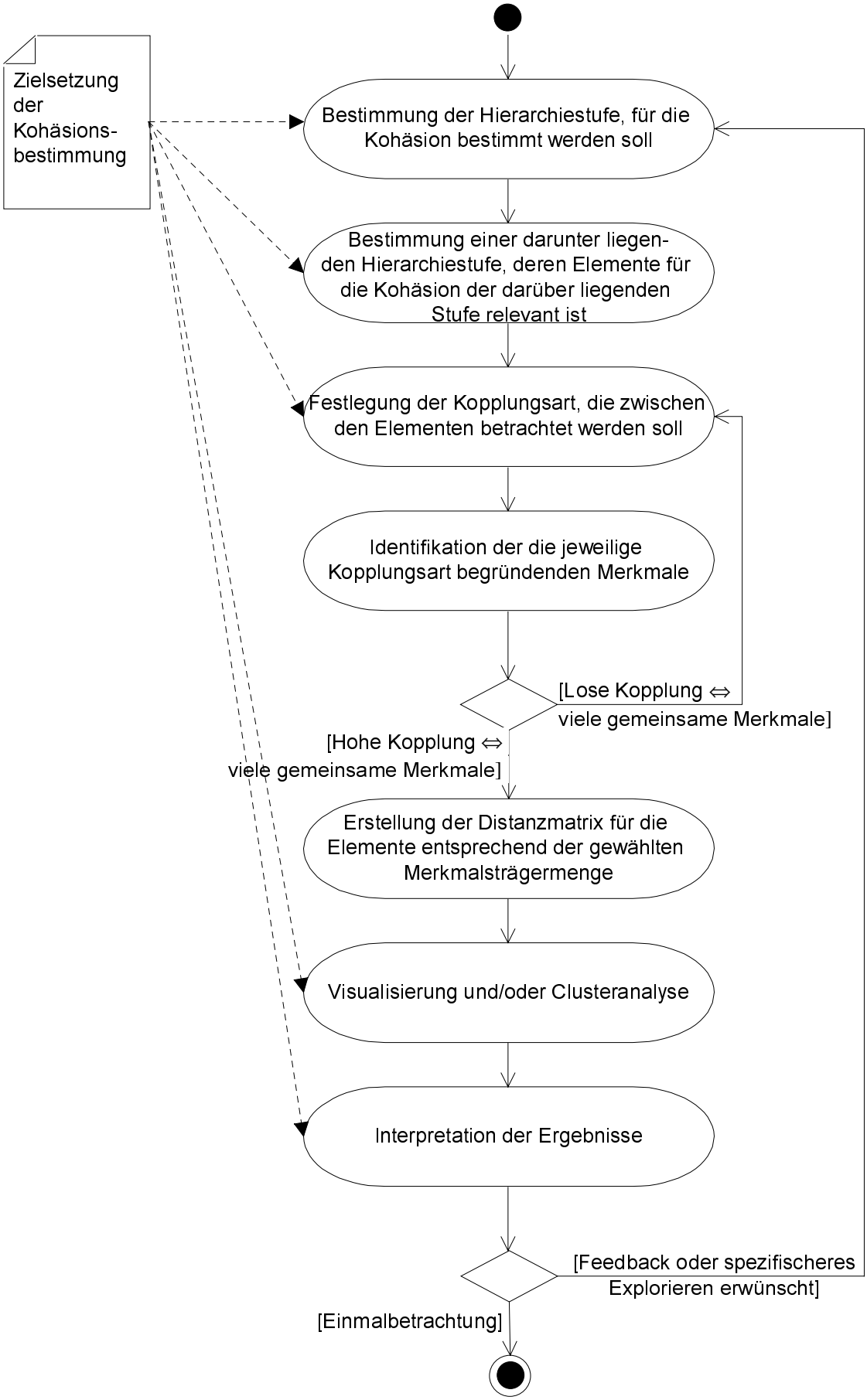


Abbildung 36: Prozeß zur kopplungbasierten Kohäsionsbetrachtung mittels des generischen Distanzmaßes

Im Vorfeld einer Kohäsionsbestimmung muß deren Zielsetzung explizit gemacht werden: Ein Großteil der späteren Aktivitäten hängt von dieser Zielsetzung und des dadurch motivierten Meßprozesses ab (vgl. Kapitel 3.3). Die danach durchzuführenden und im Diagramm angegebenen Aktivitäten bedeuten dabei im einzelnen:

1. *Bestimmung der Hierarchiestufe, für die Kohäsion bestimmt werden soll*: Je nach Ziel der Kohäsionsbestimmung werden unterschiedliche Systemhierarchiestufen von Interesse sein. Während z.B. bei einer detaillierten Kohäsionsbestimmung mit dem Zweck der Reengineering-Unterstützung durchaus einzelne, bereits im Vorfeld selektierte Funktionen oder Prozeduren im Zentrum des Interesses stehen, wird für eine qualitative Einschätzung eines großen Gesamtsystems die wesentlich höher angesiedelte Paket- bzw. Modulebene relevant sein. Innerhalb der später zu erstellenden Visualisierung oder der durchzuführenden Clusteranalyse stellt diese Hierarchiestufe dann die zu betrachtenden Entitäten zur Verfügung.
2. *Bestimmung einer darunter liegenden Hierarchiestufe, deren Elemente für die Kohäsion der darüber liegenden Stufe relevant sind*: In diesem Schritt muß festgelegt werden, welche Elemente der zu betrachtenden Entitäten die Kohäsion bestimmen. Wie in Abschnitt 7.1 gezeigt, wird hierfür i.d.R. die nächst feingranularere Hierarchiestufe verwendet, d.h. für eine Kohäsionsbestimmung von Klassen werden z.B. die Methoden betrachtet, für eine Kohäsionsbestimmung von Methoden werden z.B. die Daten-Token verwendet usw. Es ist aber durchaus möglich, auch tiefer liegende Hierarchiestufen zu verwenden, d.h. z.B. für eine Kohäsionsbestimmung von Klassen die Daten-Token der Methoden (vgl. Abschnitt 7.1.2).
3. *Festlegung der Kopplungsart, die zwischen den Elementen betrachtet werden soll*: In diesem Schritt muß festgelegt werden, welche Art von Kopplung für die Kohäsion bildenden Elemente betrachtet werden soll. Diese kann nicht unabhängig von der Art der Elemente gewählt werden: Manche Kopplungsarten erwarten eine Mindest-Hierarchiestufe, ab der sie erst sinnvoll eingesetzt werden können. So macht es z.B. wenig Sinn, die Vererbungskopplung auf der Ebene der Daten-Token einzelner Methoden zu betrachten. Statt dessen wird für diese Kopplungsart üblicherweise mindestens die Hierarchiestufe der Methoden und Attribute vorausgesetzt, da ihre vererbungbasierte Kopplung zu anderen Klassen, in denen sie z.B. definiert oder von denen sie überschrieben werden, Aufschluß über die vererbungbasierte Kohäsion zwischen Klassen geben kann. Die Anwendung der verschiedenen Kopplungsarten auf höher liegenden Hierarchiestufen kann durch das Hochkopieren relevanter Kopplungen erreicht werden, d.h. eine Entität besitzt alle Eigenschaften der in ihr enthaltenen Elemente. So ist es dadurch z.B. möglich, Vererbungskopplung auf Subsystemebene zu betrachten: Ein Subsystem $\mathcal{S}_{sub}$ erbt dabei jeweils von einem Subsystem $\mathcal{S}_{super}$, wenn es mindestens eine Klasse in $\mathcal{S}_{sub}$ gibt, die von mindestens einer Klasse aus $\mathcal{S}_{super}$ erbt. Auf dieselbe Art und Weise ist es möglich, daß sich Dateien verwenden, d.h. eine Interaktionskopplung aufweisen oder daß Subsysteme eine Komponentenkopplung aufweisen (vgl. auch Kapitel 5.3.2).
4. *Identifikation der die jeweilige Kopplungsart begründenden Merkmale*: Dieser Schritt verlangt ähnliche Kreativität, wie dies bei der Nachbildung existierender Kohäsionsmaße der Fall ist (vgl. Abschnitt 7.1). Häufig ist es hilfreich, in einem ersten Schritt exemplarisch die zu einem Element gekoppelten Elemente als Merkmale aufzufassen und darauf aufbauend die jeweilige Menge gemeinsamer Merkmale mit anderen Elementen zu notieren. Gilt in jedem Fall, daß eine entsprechend der gewünschten Kopplungsart identifizierte starke Kopplung äquivalent zu einer großen Menge gemeinsamer Merkmale ist, d.h. daß die Ordinalskala zwischen beliebigen Paaren von Entitäten für die ordinalskalierte Intervallmessung erfüllt ist (vgl. Kapitel 6.3.1), kann die allgemeine Beschreibung der Merkmalsträgermenge angegangen werden. Beispiele sind z.B. die Menge aller Attribute, die von Methoden benutzt werden können, die Menge aller Methoden, die von Methoden benutzt werden können oder die Menge aller Klassen, von denen etwas geerbt werden kann.

5. *Erstellung der Distanzmatrix für die Elemente entsprechend der gewählten Merkmalsträgermenge*: Dieser Schritt ist leicht automatisierbar und ist selbst für eine große Menge betrachteter Entitäten mit einer großen Merkmalsträgermenge auf einem normalen PC effizient durchführbar. Das Ergebnis dieser Aktivität ist die symmetrische Distanzmatrix (vgl. Kapitel 6.4).
6. *Visualisierung und/oder Clusteranalyse*: Je nach Zielsetzung der Kohäsionsbestimmung kann die im vorigen Schritt erstellte Distanzmatrix in unterschiedlicher Weise visualisiert und/oder mittels automatisch berechneter Cluster in Gruppen aufgeteilt werden. Die möglichen Freiheitsgrade bei der Visualisierung liegen in den unterschiedlichen visuellen Parametern der dargestellten Objekte (Farbe, Form, Größe); die Distanz zu den anderen Objekten ist durch Anwendung der Hauptkomponentenanalyse oder des Spring-Embedder-Verfahrens bestimmt. Die Freiheitsgrade der Clusteranalyse beschränken sich auf die mögliche Eingabe der gewünschten Anzahl zu errechnender Cluster und auf die Angabe des konkreten Clustering-Verfahrens.
7. *Interpretation der Ergebnisse*: Dieser Arbeitsschritt hängt ganz wesentlich von der Zielsetzung der Kohäsionsbestimmung ab. Die Ergebnisse der Visualisierung und/oder der Clusteranalyse sollen hierbei als wertvolle, zusätzliche Hilfe aufgefaßt werden. Je nach dem der Zielsetzung zugrundeliegenden Prozeß kann eine erneute Kohäsionsbestimmung sinnvoll sein. Für den Prozeß des Reverse-Engineering (vgl. Kapitel 3.3.3) besteht z.B. die Möglichkeit, die Kohäsionsbestimmung mittels des Top-Down-Ansatzes von der obersten Hierarchiestufe bis hin zur niedrigsten Hierarchiestufe iterativ zu verfeinern, um auf diese Art ein detailliertes Verständnis der einzelnen Komponenten zueinander zu bekommen. Für den Prozeß der Trendanalyse-Unterstützung (vgl. Kapitel 3.3.4) kann es notwendig sein, die Kohäsionsbestimmung nach Einarbeitung einer Änderung erneut durchzuführen, um eine Rückkopplung von der Änderung innerhalb der überarbeiteten Version zu erhalten.

Dieser vorgestellte Prozeß zur kopplungbasierten Kohäsionsbetrachtung mittels des generischen Distanzmaßes wird in den Kapiteln 8 und 9 für verschiedene Zielsetzungen angewendet, wobei der Schwerpunkt auf der Kohäsionsbestimmung objektorientierter Systeme liegt.

Da sich Kohäsion nicht auf Kopplung beschränken muß, wird im folgenden ein allgemeiner Prozeß zur distanzmaßbasierten Kohäsionsbestimmung vorgestellt. Das in diesem Abschnitt beschriebene Verfahren kann als Spezialisierung davon angesehen werden.

7.3.2 Allgemeiner Prozeß zur distanzmaßbasierten Kohäsionsbestimmung

Die für die Kohäsionsbestimmung relevante *Art der Zusammengehörigkeit* muß nicht auf Kopplung beschränkt bleiben. Alle Merkmale einer Entität, die in irgendeiner Weise für das Ähnlichkeitskonzept herangezogen werden können, lassen sich für eine spezielle Kohäsionsbetrachtung verwenden. Für die einzelnen in Kapitel 4.1.2 vorgestellten Merkmalsklassen läßt sich das Vorgehen zur Ableitung eines Ähnlichkeitskonzepts auf Grundlage von Merkmalen wie in der folgenden Tabelle dargestellt skizzieren.

Merkmalsklasse	Auftreten bei einer Entität	Ableitbares Kohäsionskonzept
Nominalskaliertes, binäres Merkmal.	Merkmal vorhanden oder nicht vorhanden.	• Entitäten, die das Merkmal besitzen, gehören zusammen. • Paare von Entitäten, bei denen die eine das Merkmal besitzt und bei der anderen das Merkmal nicht vorhanden ist, gehören nicht zusammen. • Entitäten, die das Merkmal nicht besitzen, gehören zusammen.
Nominalskaliertes, mehrstufiges Merkmal	Merkmal kann in verschiedenen, ungeordneten Ausprägungen vorkommen.	• Entitäten, die das Merkmal in derselben Ausprägung besitzen, gehören zusammen. • Paare von Entitäten, bei denen das Merkmal in unterschiedlichen Ausprägungen vorhanden ist, gehören nicht zusammen.
Ordinalskaliertes Merkmal	Merkmal kann in verschiedenen, geordneten Ausprägungen vorkommen; die Abstände zwischen den verschiedenen Ausprägungen sind ohne Bedeutung.	• Entitäten, die das Merkmal in derselben Ausprägung besitzen, gehören zusammen. • Paare von Entitäten, bei denen das Merkmal in unterschiedlichen Ausprägungen vorhanden ist, gehören nicht zusammen. • Entitäten gehören um so weniger zusammen, je mehr andere Entitäten bzgl. der geordneten Merkmale zwischen ihnen liegen.[22]
Intervallskalierte Merkmale	Merkmal kann in verschiedenen, geordneten Ausprägungen vorkommen; zudem sind die Abstände zwischen den verschiedenen Ausprägungen interpretierbar.	• Entitäten, die das Merkmal in derselben Ausprägung besitzen, gehören zusammen. • Entitäten gehören um so weniger zusammen, um so größer der Abstand bzgl. des Merkmals ist.

[22] Der Verzicht auf eine graduelle Zusammengehörigkeit ist hier notwendig, um die Visualisierung und die Clusteranalyse valide zu halten. Da keinerlei Aussagen über die Intervalle zwischen den geordneten Merkmalen vorliegen, diese aber bei beiden Techniken dargestellt bzw. verwendet werden, muß die Restriktion auf ordinale Aussagen während der Interpretation der Ergebnisse permanent berücksichtigt werden.

Verhältnisskalierte Merkmale	Merkmal kann in verschiedenen, geordneten Ausprägungen vorkommen. Darüber hinaus sind die Verhältnisse zwischen den Merkmalsausprägungen definiert und interpretierbar.	• Entitäten, die das Merkmal in derselben Ausprägung besitzen, gehören zusammen. • Entitäten gehören um so weniger zusammen, um so größer der Abstand bzgl. des Merkmals ist.[23]

Wird für die Kohäsionsbetrachtung nicht nur ein Merkmal, sondern eine Menge von Merkmalen verwendet, so ist zu berücksichtigen, daß das ableitbare Kohäsionskonzept von der Merkmalsklasse mit den schwächsten Informationen abhängt. Insgesamt sind solche Misch-Merkmalsklassen selten sinnvoll, da ihnen i.d.R. die semantische Geschlossenheit für die spätere Interpretation fehlt. Werden allerdings mehrere Merkmale einer Merkmalsklasse für die Kohäsionsbetrachtung verwendet, erlaubt in einem solchen Fall die Analyse zusätzliche Interpretationen über die Ähnlichkeit bzgl. der Merkmalsmenge: So sind ebenfalls die sich bildenden Gruppierungen von Entitäten interpretierbar, die bzgl. aller betrachteten Merkmale identische Ausprägungen besitzen (vgl. z.B. die identifizierten Gruppierungen in Abbildung 32).
Die Verwendung der ersten drei Merkmalsklassen für eine Kohäsionsbetrachtung entspricht hierbei der in Kapitel 6.3.1 vorgestellten ordinalskalierten Kantenvermessung, wohingegen die letzten zwei Merkmalsklassen in Kapitel 6.3.2 als intervallskalierte Kantenvermessung bezeichnet werden.
Typische Beispiele solcher nicht kopplungbasierten Kohäsionsbetrachtungen für Softwaresysteme sind z.B.:

- *Dateiextension* als nominalskaliertes, mehrstufiges Merkmal: Hiermit ist es möglich, einen schnellen Überblick über die verschiedenen vorkommenden Dateitypen und deren Häufigkeiten innerhalb eines Softwaresystems zu erhalten. Dateien mit identischer Extension bilden sich dabei sowohl innerhalb der Visualisierung als auch während der Clusteranalyse deutlich heraus.
- *Autorenschaft von Dateien* als nominalskaliertes, mehrstufiges Merkmal: Hiermit ist es möglich, einen schnellen Überblick über die verschiedenen Autoren eines Softwaresystems und ihre jeweiligen Produkte zu erhalten. Dateien vom selben Autor bilden sich dabei sowohl innerhalb der Visualisierung als auch während der Clusteranalyse deutlich heraus.
- *Letztes Änderungsdatum von Dateien* als intervallskaliertes Merkmal: Hiermit ist es möglich, einen schnellen Überblick über die verschiedenen Zeitpunkte zu erhalten, zu denen Produkte in letzter Zeit geändert wurden. Je größer der Zeitunterschied zwischen zwei Dateien, desto weniger gehören sie bzgl. des letzten Änderungsdatums zusammen. Dateien, die gemeinsam (genauer: am selben Datum) geändert wurden, bilden sich dabei sowohl innerhalb der Visualisierung als auch während der Clusteranalyse deutlich heraus.

[23] Die Verschärfung, daß der Abstand zwischen den Entitäten dem Verhältnis der Merkmale entspricht, ist für die Clusteranalyse ebenfalls anwendbar. Die Visualisierung solcher Entitäten ist i.d.R. allerdings nicht entsprechend möglich, da i.A. bei der Visualisierung rationaler Daten aufgrund der Dimensionsreduktion Informationen in der Visualisierung verloren gehen (vgl. Kapitel 6.4.2); dieser Verlust ist bei ordinalen Daten, bei denen die Abstände unter Einhaltung der Reihenfolge frei variiert werden können, deutlich geringer als bei verhältnisskalierten Daten, bei denen zusätzlich die Intervall-Reihenfolge beibehalten werden muß.

- *Meßwerte* als verhältnisskalierte Merkmale: Wie bereits in Kapitel 6.3.2 erläutert ist es möglich, Meßwerte als verhältnisskalierte Merkmale aufzufassen (bei entsprechendem Skalenniveau). Bzgl. des Maßes LOC gehören demnach Dateien zusammen, deren LOC Wert ähnlich ist. Innerhalb der Visualisierung und der Clusteranalyse ist damit sehr einfach die Bestimmung von Extremwerten möglich (z.B. extrem hoher LOC-Wert).

Der Prozeß zur allgemeinen Kohäsionsbestimmung mittels des generischen Distanzmaßes ist recht ähnlich zum Prozeß der kopplungbasierten Kohäsionsbestimmung und wiederum in der Notation der Aktivitätendiagramme dargestellt (vgl. Abbildung 37).
Nach der Bestimmung der Hierarchiestufe, für die Kohäsion bestimmt werden soll und derjenigen, deren Elemente für die Kohäsion relevant ist (vgl. Abschnitt 7.3), werden die Merkmale bestimmt, die die Zusammengehörigkeit zwischen den Elementen begründen sollen. Wichtig ist hier, daß diese Merkmale bzgl. der Merkmalsklassifizierung eindeutig zugeordnet werden können, da die Klasse relevant für die spätere Interpretation der Ergebnisse ist (vgl. Abschnitt 7.3.2): Während die Entdeckung von Clustern für nominalskalierte Merkmale lediglich bzgl. der Zugehörigkeit einzelner Entitäten zu einer Gruppe oder bzgl. der Gruppengröße interpretierbar sind, können Cluster für mindestens ordinalskalierte Merkmale geordnet werden, erlauben also daher Aussagen über Gruppen von Entitäten. Die Mächtigkeit dieser Aussagen nimmt mit den intervall- und verhältnisskalierten Merkmalen noch weiter zu. Der oben angedachte Fall, daß mehrere Merkmale aus verschiedenen Merkmalsklassen gleichzeitig berücksichtigt werden sollen, wird hier nicht unterstützt, da solche Fälle praktisch kaum vorkommen. In solchen Fällen ist die separate Kohäsionsbetrachtung für jede einzelne Merkmalsklasse empfehlenswert.

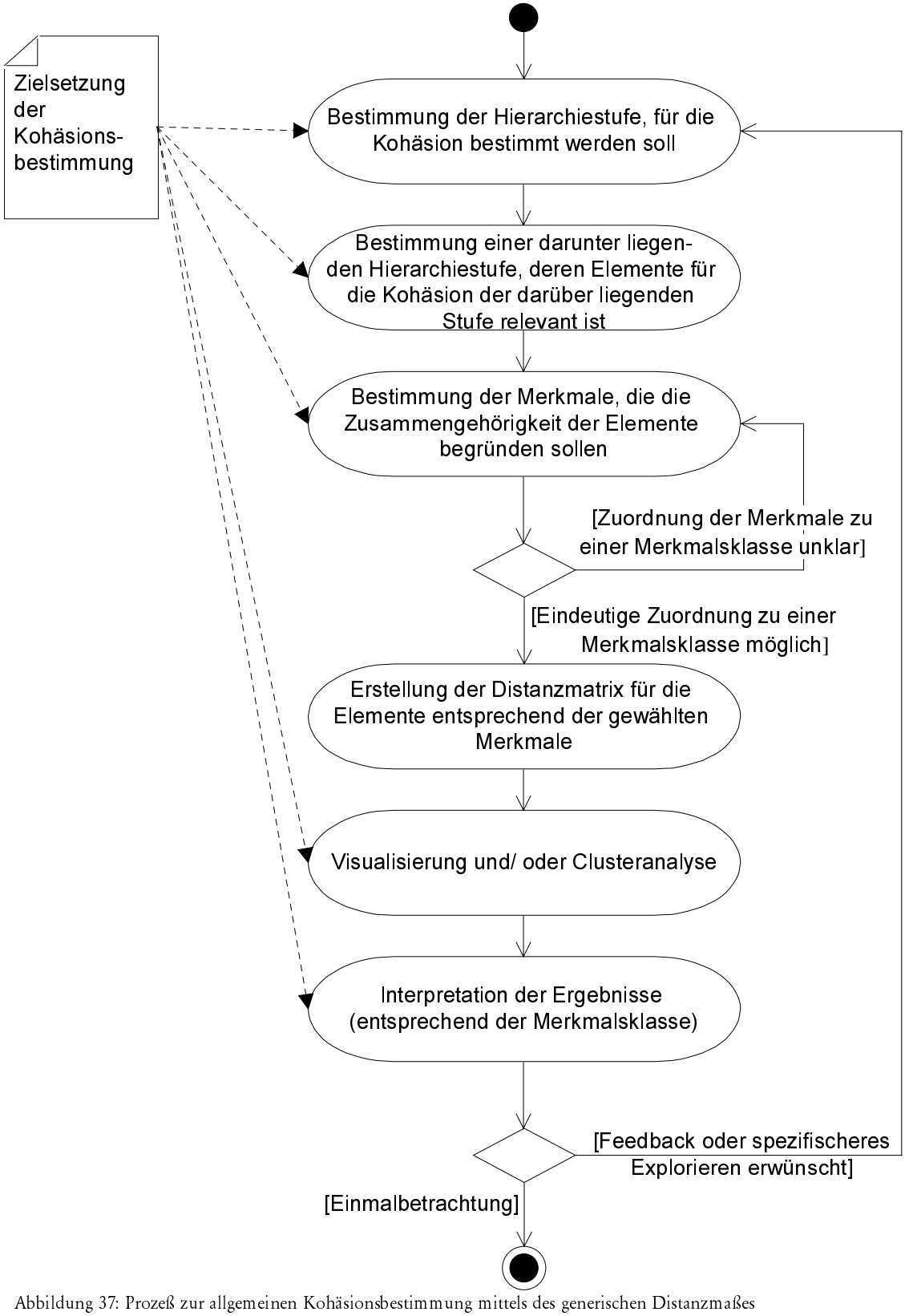


Abbildung 37: Prozeß zur allgemeinen Kohäsionsbestimmung mittels des generischen Distanzmaßes

Diese Form der allgemeinen Kohäsionsbestimmung wird z.B. in einem Beispiel in Kapitel 8.5 angewendet, in dem es um die Zusammengehörigkeit von Dateien bzgl. ihrer Autorenschaft geht.

7.4 Zusammenfassung

Kohäsion und Kopplung sind wesentliche Qualitätskriterien innerhalb der Softwarebearbeitung. Für beide Konzepte gibt es unterschiedlichste Sichtweisen, die sich in unterschiedlichsten Softwaremaßen zur quantitativen Bestimmung der graduellen Ausprägung von Kohäsion und Kopplung äußern. Mittels des Konzepts des generischen Distanzmaßes ist es sowohl möglich, jeweils die unterschiedlichen Konzepte mit einer einheitlichen Nomenklatur und einem einheitlichen Vorgehen nachzubilden, als auch die beiden Konzepte miteinander zu vereinen. Letzteres ist möglich, da Kohäsion in der Form, wie sie bisher verwendet wurde, für eine Entität als Bewertung der Zusammengehörigkeiten der in ihr vorkommenden Elemente bestimmt wird, d.h. zwei der drei eine spezielle Kohäsionssicht aufspannenden Dimensionen sind die *Art der zu betrachtenden Entität* und die *Art der enthaltenden Elemente*. Dieses Muster der Kohäsionsbestimmung einer Entität mittels der in ihr enthaltenen Elemente findet sich in allen bisherigen kopplungbasierten Kohäsionsbetrachtungen wieder. Das Zusammenspiel zwischen Kohäsion und Kopplung wird in der dritten Dimension, der *Art der Zusammengehörigkeit der Elemente* deutlich: Kopplung, die den Grad der Abhängigkeit zwischen mehreren Elementen betrachtet, kann eine spezielle Form der Zusammengehörigkeit zwischen Elementen darstellen, d.h. durch die Bewertung auf einer höheren Hierarchiestufe Kohäsion begründen. Eine hohe Kopplung zwischen zwei Entitäten wird demnach als eine hohe Zusammengehörigkeit aufgefaßt; besitzen alle Elemente einer bzgl. Kohäsion zu untersuchenden Entität eine hohe Kopplung, so wird die Kohäsion dieser Entität als hoch angesehen.
Sowohl die bisherigen Kohäsionsansätzen zugrundeliegenden Zugehörigkeitsarten als auch die der Kopplung zugrundeliegenden Abhängigkeitsarten können mittels des generischen Distanzmaßes nachgebildet werden: es ist damit als generische Technik zur kopplungbasierten Kohäsionsbestimmung einsetzbar.
Darüber hinaus kann Kohäsion auch durch nicht-kopplungbasierte Zusammengehörigkeiten begründet sein: Je nach Zugehörigkeit der betrachteten Merkmale zu einer der fünf Merkmalsklassen kann die Anwendung des generischen Distanzmaßes zu unterschiedlich zu interpretierenden Ergebnissen führen. Kohäsion kann damit als generisches Qualitätsmerkmal den unterschiedlichsten Zielsetzungen der unterschiedlichen Meßprozesse optimal angepaßt werden und mittels des generischen Distanzmaßes meßbasiert analysiert werden.

„Mithin können sie [die Begriffe, Anm.d.A.]
durch eine empirische Anschauung belegt, d.h. der Gedanke davon
an einem Beispiele gewiesen (demonstriert, aufgezeigt) werden;
und dieses muß geschehen können: widrigenfalls man nicht gewiß ist,
ob der Gedanke nicht leer, d.h. ohne alles Objekt sei.“
(Kant in „*Kritik der Urteilskraft*“,
§57: „*Auflösung der Antinomie des Geschmacks*“)

8 Kohäsion objektorientierter Systeme

Die im vorigen Kapitel erarbeiteten Konzepte zur Kohäsionsbestimmung sollen in diesem Abschnitt auf *objektorientierte Systeme* angewendet werden. Das objektorientierte Vorgehen mit seinen neuen Konzepten (s.u.) stellt eine vielversprechende Möglichkeit dar, das Software-Engineering (vgl. Kapitel 3) strukturierter zu gestalten und die Qualität der ausgelieferten Produkte (im Sinne der internen Qualität, vgl. Kapitel 2) zu erhöhen.
Die grundlegenden Konzepte objektorientierter Systeme sind:

- *Datenkapselung (Information Hiding)*, d.h. das Anbieten von Operationen auf extern nicht sichtbaren Strukturen. So sieht das Klassenkonzept vor, daß die intern sichtbaren Attribute, die den Datenbestand repräsentieren und für die die Klasse verantwortlich ist, nur mittels explizit dafür vorgesehener Methoden verwendet werden dürfen.
- *Vererbung (Inheritance)*, d.h. die Möglichkeit, zwischen verschiedenen Klassen eine Generalisierungs-/Spezialisierungs-Relation bzw. eine Schnittstellen-/Implementierungs-Relation zu definieren (vgl. dazu auch Abschnitt 8.2). Die Unterklasse verhält sich dabei wie die Oberklasse, hat aber die Möglichkeit, geerbtes Verhalten zu modifizieren und zu erweitern.
- *Polymorphie (Polymorphism)*, d.h. die Möglichkeit, die Zuordnung einer Nachricht zu einer konkreten Implementierung erst zur Laufzeit vorzunehmen. Ein und dieselbe Nachricht kann damit zur Laufzeit des Systems unterschiedliche Implementierungen verwenden.

Auch wenn einige dieser Konzepte ebenfalls in nicht objektorientierten Sprachen möglich sind (z.B. Datenkapselung mittels eines konsequent eingesetzten Modulkonzepts, vgl. z.B. [PaSi94]), so erlauben objektorientierte Programmiersprachen, diese Konzepte mittels eigener Sprachkonstrukte explizit zu machen. Dies ist speziell für die statische Analyse der Quelltexte mit dem Ziel der Qualitätssicherung relevant, da explizit gemachte Strukturen sehr einfach automatisch zu extrahieren sind. Das der Vermessung zugrundeliegende und mittels Werkzeugen zu erstellende Produktmodell (vgl. Kapitel 3.2) kann damit mehr Informationen beinhalten und erlaubt eine Vermessung bzgl. vielfältigerer Aspekte.
Daß diese Form der Qualitätssicherung auch innerhalb objektorientierter Systeme notwendig ist, hat sich aus der Erkenntnis der mittlerweile langjährigen Erfahrung mit derartigen Systemen gezeigt, die besagt, daß das von Lehman und Belady formulierte zweite Gesetz der Softwareevolution, das *Gesetz der zunehmenden Entropie (Law of Increasing Entropy)* [LeBe85], auch für diese neue Art von Systemen gültig ist:

> *Gesetz der zunehmenden Entropie*: Die Entropie eines Systems, d.h. dessen Fehlen an Wohlstrukturiertheit, wächst ab dessen ersten Erstellung mindestens so lange, bis spezielle Aktivitäten zu deren Beseitigung vorgenommen werden.

Definition 35: Gesetz der zunehmenden Entropie (vgl. [GhJaMa91], S. 412)

Da ein wesentliches Ziel der erhöhten Softwarequalität der langjährige Einsatz und damit verbunden die Notwendigkeit der permanenten Wartung ist, kompensiert sich für OO-Systeme die verlangsamte Geschwindigkeit der Entropie mit dieser erhofften Langlebigkeit von OO-Systemen. Daher wird in jüngster Zeit der Begriff des *Altsystems (Legacy System)* auch schon für einige objektorientierte Systeme verwendet.
Kohäsion als ein generisches Konzept (vgl. Kapitel 7) kann aufgrund der zusätzlich vorhandenen Strukturdaten objektorientierter Systeme auf unterschiedlichen Hierarchieebenen begutachtet werden (vgl. Schritt 1 – Bestimmung der Hierarchiestufe, für die Kohäsion bestimmt werden soll – in Kapitel 7.3 und 7.4), von denen die wichtigsten Ebenen in Abschnitt 8.1 vorgestellt werden. Vor konkreten Anwendungen des Prozesses muß jedoch noch das für OO-Systeme essentielle Konzept der Vererbung analysiert werden, da anderenfalls die Kohäsionsbetrachtung an Information verlieren würde. In Abschnitt 8.2 wird daher ein Konzept vorgestellt, das die Vererbung in die Kohäsionsbetrachtung mit einfließen läßt. Damit werden die bei der Anwendung vieler typischer Kohäsionsmaße (und von Maßen im allgemeinen, vgl. [SiBeLe01]) unscharfen Aussagen bzgl. des Umgangs mit geerbter Funktionalität thematisiert, für die weitere Arbeit detaillierte Lösungen erarbeitet und damit das Vererbungs-Meßwerte-Problem (vgl. Kapitel 3.6.2) vollständig gelöst.
Für die kopplungbasierte Kohäsion werden entsprechend des Prozesses aus Kapitel 7.3.1 anschließend fünf Anwendungen in Abschnitt 8.3 vorgestellt, die als Hierarchiestufe, deren Elemente für die Kohäsion der darüber liegenden Stufen relevant sind, die Klassenebene verwenden, d.h. z.B. für eine Kohäsionsbetrachtung auf der Subsystemebene verwendet werden können. Als Kopplung zwischen Klassen wird in Abschnitt 8.3.1. Vererbung und in Abschnitt 8.3.2. Interaktion verwendet. In Abschnitt 8.4. wird als Hierarchiestufe, deren Elemente für die Kohäsion der darüber liegenden Stufen relevant sind, die Methoden- und Attributsebene verwendet.
Für die allgemeine Form der merkmalbasierten Kohäsionsbetrachtung entsprechend des Prozesses aus Kapitel 7.3.2 wird in Abschnitt 8.5 eine typische Anwendung vorgestellt, innerhalb derer die Autorenschaft von Dateien als Merkmale extrahiert werden.

8.1 Abstraktionsniveaus objektorientierter Systeme

Ein wichtiges Mittel für die Beherrschung von Komplexität ist die Erstellung von geeigneten Abstraktionsniveaus, von denen aus ein System (in diesem Fall ein objektorientiertes) betrachtet werden kann. Die einfachste Form der Kriterien, die dafür verwendet werden können, sind die „*ist-Teil-von*"- bzw. die „*besteht-aus*"-Relationen, da diese Relationen direkt dem System entnehmbar sind, aufgrund der einfachen und intuitiven Übertragung auf physische Objekte (z.B. Festplatten, Verzeichnisse, Files etc.) vom Entwickler i.d.R. auch bewußt verwendet werden und zudem während der Analyse und Entwurfsphase während der permanenten Dekomposition in Teilprobleme abfallen. Darüber hinaus stellen diese Abstraktionsniveaus bei allen Produktmodellen diejenigen Entitäten zur Verfügung, die für eine spätere Vermessung in Frage kommen (vgl. Kapitel 5).
Für OO-Systeme sind unterschiedliche Gruppen von Abstraktionsniveaus etabliert:

- Innerhalb des *Multidimensional Framework (MDF)* für objektorientierte Maße von Hendersson-Sellers werden innerhalb der dort aufgezeigten dritten Dimension folgende Abstraktionsniveaus angegeben ([Hend96], S. 61):
  - *System* (gesamtes, betrachtetes OO-System),
  - *Subsysteme* (allgemeine Aufteilung der Klassen zu Paketen),
  - *Klassen* (als atomare Einheit),
  - *Services* (für die von den Klassen angebotene, sichtbare Funktionalität),
  - *Methoden* (für die innerhalb einer Klasse sichtbare Funktionalität).

- Innerhalb einer allgemeinen Dekomposition objektorientierter Systeme unterscheidet Meyer folgende Abstraktionsniveaus ([Meye97]):
  - *System* (gesamtes, betrachtetes OO-System),
  - *Cluster* (Aufteilung in Beziehung stehender Klassen zu Paketen),
  - *Klassen* (als atomare Einheit),
  - *Feature* (für die innerhalb einer Klasse definierten Methoden und Attribute).
- Innerhalb der Kopplungsbetrachtung von Eder, Kappel und Schrefl (vgl. Kapitel 7.2) werden folgende Abstraktionsniveaus vorgeschlagen ([EdKaSc93]):
  - *Vererbungshierarchien* (Klassen, die in einer Vererbungsbeziehung zueinander stehen),
  - *Klassen* (als atomare Einheit, dort Komponente, vgl. Kapitel 7.2)
  - *Methoden* (für die innerhalb einer Klasse definierten Operationen).
- In Abhängigkeit von der jeweiligen Softwareentwicklungsphase unterscheidet Rüping jeweils mehrere Ebenen [Rüpi94]. Für das Design sind dies z.B. die folgenden Abstraktionsniveaus:
  - *Subsysteme* (allgemeine Aufteilung der Klassen zu Paketen),
  - *Verträge* (als Regeln für das Zusammenspiel der einzelnen Klassen),
  - *Klassen* (als Implementierung zur Erfüllung der Verträge).
- Innerhalb der verschiedenen Ebenen für Softwarevermessung objektorientierter Systeme unterscheiden Briand, Daly und Wüst folgende Abstraktionsniveaus ([BrDaWü99]):
  - *System* (gesamtes, betrachtetes OO-System),
  - *Klassenmengen* (allgemeine Aufteilung der Klasse zu Paketen),
  - *Klassen* (als atomare Einheit),
  - *Methoden* (als Implementierung der von der Klasse bereit gestellten Funktionalität),
  - *Attribute* (als den Methoden zugrundeliegender Datenbestand).

Alle diese vorgeschlagenen Niveaus, von denen die meisten als Teilmenge der in Kapitel 5.3.2 hergeleiteten Hierarchiestufen betrachtet werden können, sind potentielle Betrachtungsebenen für Kohäsion, d.h. mögliche Ergebnisse des Arbeitsschritts 1 innerhalb des Prozesses aus Kapitel 7.3.1 bzw. Kapitel 7.3.2 zur Bestimmung der Hierarchiestufe, für die Kohäsion bestimmt werden soll.

Statt die Kohäsionsbestimmung für jede dieser Hierarchiestufen explizit durchzuführen, werden im folgenden diejenigen Hierarchiestufen betrachtet, deren Elemente mit ihren Kopplungen und Merkmalen für die Kohäsionsbetrachtung relevant sein können (Schritt 2 aus Kapitel 7.3 und 7.4). Entsprechend des Kohäsionsmodells in Kapitel 7.3.1 kann jede dieser Hierarchiestufen dann für die Kohäsionsbetrachtung aller über ihr liegenden Hierarchiestufen verwendet werden.
Direkte Kopplungen bzw. die für eine Kohäsionsbetrachtung relevanten Merkmale sind für folgende Hierarchiestufen möglich und können daher jeweils für eine Kohäsionsbetrachtung aller darüber liegenden Hierarchiestufen verwendet werden:

- *Subsysteme*: In einigen Programmiersprachen wie z.B. Java können Kopplungen zwischen Subsystemen direkt durch allgemeine *import*-Anweisungen (z.B. `import <Subsystem>.*`) erzeugt werden.
- *Dateien*: Direkte Kopplungen sind durch die *include*- bzw. *import*-Anweisungen gegeben. Mögliche Merkmale sind z.B. Dateityp (z.B. Header-Dateien vs. Implementierungsdateien), Dateiautor, oder letztes Änderungsdatum der Datei.
- *Klassen*: Direkte Kopplungen sind vor allen Dingen durch die Komponentenkopplung und Vererbungskopplung gegeben (vgl. Kapitel 7.2.2 und 7.2.3). Hierbei ist eine Klasse binär mit anderen Elementen gekoppelt, d.h. eine Klasse ist Oberklasse einer anderen Klasse bzw. ein Benutzen einer Methode einer anderen Klasse von beliebig vielen eigenen Methoden wird als eine binäre Benutzung der Klasse aufgefaßt. Diese Form der interaktionbasierten

Kopplungsvereinfachung, innerhalb derer Klassen binär gekoppelt sind, wenn mindestens ein Klassenelement der einen Klasse mit mindestens einem Klassenelement der anderen Klasse eine Interaktionskopplung besitzt, findet auch in Klassendiagrammen Anwendung, in denen ebenfalls die Klasse als atomare Einheit Kopplungen besitzt. Eine feingranularere Betrachtung von Kopplung zwischen Klassen verfeinert die einfache Benutzung einer Klasse um die Menge ihrer aufgerufenen Methoden. Dadurch wird eine Abstufung der Kopplung zwischen Klassen möglich (vgl. Abschnitt 8.3).
Mögliche Merkmale von Klassen sind z.B. Klassentyp (abstrakte Klasse, Interface, Implementierungsklasse, etc.), Klassenautor oder die für ihre Implementierung verwendete Programmiersprache.

- *Methoden/Attribute*: Direkte Kopplungen sind durch die Interaktionskopplung gegeben (vgl. Kapitel 7.2.1). Da Methoden und Attribute in keinerlei Enthaltensein-Relation stehen, werden beide Entitäten auf derselben Hierarchiestufe angeordnet.

Für eine kopplungbasierte Kohäsionsbetrachtung sind besonders die durch Klassen und Methoden/Attribute gegebenen Kopplungen interessant, da diese auch explizit innerhalb des Softwareentwurfs erstellt werden. Die Kopplungen zwischen Dateien und Subsystemen[24] entstammen eher dem Bereich der Implementierung, bei der es darum geht, den Entwurf auf physische Elemente (Dateien, Verzeichnisse) abzubilden.
Entsprechend des Kohäsionsmodells (vgl. Kapitel 7.3.1) können die Kohäsion begründenden Kopplungen und Merkmale dieser beiden Ebenen für jeweils alle über der jeweiligen Hierarchiestufe liegenden Ebenen verwendet werden. Im einzelnen bedeutet dies:

- Kopplungen bzw. Merkmale von Klassen können verwendet werden für
  - Dateikohäsion, d.h. inwieweit die in einer Datei implementierten Klassen zusammengehören. Diese Betrachtungssicht ergibt üblicherweise allerdings keine sehr ausdruckstarken Ergebnisse, da alle gängigen Programmierrichtlinien empfehlen, pro Datei nur eine (öffentliche) Klasse zu implementieren (vgl. z.B. [HeNy96]); in JAVA ist dies sogar zwingend vorgeschrieben (vgl. auch Kapitel 3: „*JAVA-File-Organization*“ der *JAVA-Code-Conventions* in [CWHT98]).
  - Subsystemkohäsion, d.h. inwieweit die innerhalb eines Subsystems existierenden Klassen zusammengehören. Diese Betrachtung erlaubt Aussagen darüber, inwieweit die Aufteilung der Klassen auf Subsysteme durch die Benutzungsrelationen (Komponentenkopplung der Klassen), Vererbungsrelationen (Vererbungskopplung der Klassen) oder andere betrachtete Merkmale motiviert ist.
  - Systemkohäsion, d.h. inwieweit die Klassen des Systems zusammengehören. Diese Betrachtung erlaubt Aussagen darüber, welche grundlegenden Architekturen das System besitzt. Besteht das System aus mehreren deutlich voneinander abzugrenzenden Teilsystemen (z.B. einem Client- und einem Server-Teil), so kann dies i.d.R. aufgrund der Klassenkopplungen identifiziert werden.
- Kopplungen bzw. Merkmale von Methoden/Attributen können verwendet werden für
  - Klassenkohäsion, d.h. inwieweit die in einer Klasse enthaltenen Attribute und Methoden zusammengehören. Diese Betrachtung erlaubt z.B. Aussagen darüber, ob die Klasse tatsächlich die saubere Implementierung eines abstrakten Datentyps ist, oder ob es sich um reine Hilfsklassen, Datencontainer oder andere auf der Skala für abstrakte Kohäsion suboptimale Aufteilungen handelt (vgl. Kapitel 7.1.2).

[24] Auch hier ist die Abhängigkeit von der Programmiersprache bedeutend: Während Subsysteme in C++ keinerlei Sprachäquivalent besitzen, wurde in JAVA mit den *Packages* ein Konzept mit eigenen Regeln eingeführt (z.B. Sichtbarkeiten von Klassen), das innerhalb des Softwareentwurfs identifizierte Subsysteme entsprechend umsetzt (vgl. z.B. [CWHT98]).

- Die auf Kopplung zwischen Methoden/Attributen basierende Datei-, Subsystem- und Systemkohäsion entspricht den auf Kopplung bzw. Merkmalen von Klassen basierenden Kohäsionsarten.

Bevor diese beiden für Kopplungen bzw. Merkmale herangezogenen Hierarchiestufen für die Kohäsionsbetrachung detailliert beschrieben werden, muß das für OO-Systeme essentielle Konzept der Vererbung entsprechend berücksichtigt werden (vgl. Vererbungs-Meßwert-Problem in Kapitel 3.6.2): Aufgrund der Möglichkeit, Funktionalität von Unterklassen in Oberklassen hinein zu faktorisieren und der damit gegebenen Problematik, innerhalb einer statischen Quelltextanalyse bei Klassen nur die neu implementierte Funktionalität zu berücksichtigen, die unter Umständen nur einen Bruchteil der durch diese Klasse tatsächlich verfügbaren Funktionalität darstellt, müssen für die Kohäsionsbetrachtung Klassen (und damit evtl. die über ihr liegenden Hierarchiestufen) in Vererbungsbeziehungen separat behandelt werden. Das für dieses Ziel konzipierte Konzept des *Flatten* wird im folgenden Abschnitt erläutert.

8.2 Berücksichtigung von Vererbung

Vererbung ist ein wesentliches Konzept objektorientierter Systeme und bezieht sich auf die Klassenstruktur: Ausgehend von einer Klasse C werden als *direkte Oberklassen* diejenigen Klassen bezeichnet, von denen C direkt erbt und als *direkte Unterklassen* diejenigen Klassen bezeichnet, die direkt von C erben. Die Frage, ob eine Klasse Ober- oder/und Unterklasse ist, hängt von dem gewählten Standpunkt ab. Da direkte Oberklassen einer Klasse C selbst ebenfalls direkte Oberklassen besitzen können, kann eine Klasse C neben direkten Oberklassen auch *indirekte Oberklassen* besitzen. Entsprechendes gilt für *indirekte Unterklassen*.
Die Vererbungsbeziehungen zwischen den Klassen können als gerichteter Graph dargestellt werden (da eine Klasse durchaus mehrere direkte Oberklassen besitzen kann, handelt es sich nicht zwangsläufig um einen Baum). Hierbei hat sich die Sprechweise durchgesetzt, daß die Klasse, von der geerbt wird, als *Oberklasse*, und die Klasse, die von ihr erbt, als *Unterklasse* bezeichnet wird, d.h. auf eine Klasse C, die die Sichtweise vorgibt, wird verzichtet.
Jede Kante, d.h. jede einzelne Vererbungsbeziehung zwischen einer Ober- und einer Unterklasse kann für verschiedene, nicht zwangsläufig einander ausschließende Ziele eingesetzt werden:

- *Generalisierung/Spezialisierung*: Hierbei stellt die Oberklasse eine Generalisierung ihrer Unterklasse dar. Vom Standpunkt der Oberklasse stellt die Unterklasse eine Spezialisierung dar. Die Aufgabe der Unterklasse besteht darin, diejenige Funktionalität zu implementieren, die das spezifische, über die Funktionalität der Generalisierung hinausgehende Verhalten betrifft.
- *Schnittstellenabstraktion*: Hierbei gibt die Oberklasse die Schnittstellen vor, die in der Unterklasse implementiert werden müssen.
- *Typhierarchieimplementierung*: Hierbei dient die Unterklasse der Erstellung einer Hierarchie unterschiedlicher Typen, in der die Oberklasse einen Obertypen darstellt (vgl. Kapitel 17 in [Meye97]). Beide Typen besitzen implementiertes Verhalten (im Gegensatz zur Schnittstellenabstraktion); die Signaturen der Methoden beider Klassen sind identisch (d.h. jede Funktionalität der Oberklasse ist auch in der Unterklasse verfügbar), ihre Implementierungen können aber voneinander abweichen. Das Ziel ist die gegenseitige Austauschbarkeit für eine die Typhierarchie verwendende Klasse, d.h. es soll für sie irrelevant sein, ob sie den Obertypen oder den Untertypen verwendet.
- *Quelltextwiederverwendung*: Hierbei benötigt die Unterklasse Teile des in der Oberklasse implementierten Verhaltens. Ansonsten stehen die Ober- und Unterklasse in keinerlei Verhältnis.

Auch wenn diese Vererbungsmotive nicht immer in Reinform vorliegen (so stellen abstrakte Klassen z.B. üblicherweise eine Schnittstellenabstraktion und eine Generalisierung/ Spezialisierung dar), so sollen ihre Auswirkungen auf das Kohäsionskonzept und auf klassische, knotenbasierte Softwaremaße im folgenden dennoch jeweils separat betrachtet werden (vgl. [SiBe00], [SiBeLe01]).

8.2.1 Generalisierung/Spezialisierung

Typisches Indiz der Generalisierung/Spezialisierung ist die Angabe zusätzlicher Funktionalität in der Unterklasse. Die von der Oberklasse bereitgestellte Funktionalität ist dabei ebenfalls in der Unterklasse verfügbar. Das in Abbildung 38 dargestellte Klassendiagramm in UML-Notation (vgl. [FoSc98]) soll eine typische derartige Vererbungsbeziehung darstellen:

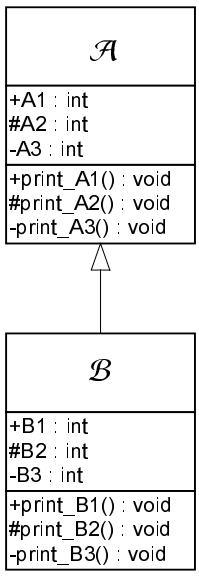


Abbildung 38: Beispiel einer Generalisierung/ Spezialisierungs-Vererbung

Objekte der Klasse $\mathcal{B}$ bieten die von $\mathcal{A}$ geerbte Funktionalität (Attribute `A1`, `A2` und Methoden `print_A1()`, `print_A2()`) plus die in $\mathcal{B}$ zusätzlich implementierten Attribute und Methoden.

Bei der Betrachtung von auf Klassenelementen basierender Klassenkohäsion (oder aller darüber liegenden Hierarchiestufen) und der damit verbundenen Frage, inwieweit die Elemente einer Klasse zusammengehören, kann die bloße Betrachtung der in $\mathcal{B}$ neu implementierten Funktionalität zu unvollständigen Ergebnissen führen: Da sich die Klasse $\mathcal{B}$ nach außen anders verhält, als die ausschließliche Betrachtung der in $\mathcal{B}$ neu definierten Funktionalität vermuten läßt, geht es bei der Kohäsionsbetrachtung nicht darum, inwieweit die neu implementierte Funktionalität zusammengehört, sondern inwieweit alle Elemente der Klasse, so wie sie auch von einer benutzenden Klasse gesehen werden, zusammengehören. Nur diese vollständige, Vererbung repräsentierende Sicht erlaubt die Betrachtung der abstrakten Kohäsion einer Klasse.

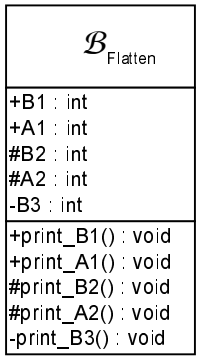


Abbildung 39: Klasse $\mathcal{B}$ aus Abbildung 38 nach dem Flatten

Ebenfalls bei der Betrachtung von auf Klassen basierender Dateikohäsion (oder aller darüber liegenden Hierarchiestufen) und der damit verbundenen Frage, inwieweit die jeweiligen Klassen zusammengehören, kann die ausschließliche Betrachtung von $\mathcal{B}$, d.h. ihrer neu implementierten Funktionalität, zu unvollständigen Ergebnissen führen: Im Extremfall, in dem $\mathcal{B}$ keinerlei Funktionalität hinzufügt, würde das auf Komponentenkopplung basierende Kohäsionskonzept eine Unähnlichkeit zu allen anderen Klassen liefern, da $\mathcal{B}$ keinerlei Interaktionen nach außen besitzt. Besitzt die Oberklasse $\mathcal{A}$ allerdings eine diesbezügliche Ähnlichkeit zu anderen Klassen, so liegt ein Bruch in der Betrachtung vor: Obwohl sich $\mathcal{A}$ und $\mathcal{B}$ nach außen vollständig identisch verhalten (sie sind von der Funktionalität her gleich), besitzen sie stark unterschiedliche Ähnlichkeiten zu anderen Klassen.

Diese Problematik läßt sich auf alle Hierarchiestufen erweitern, deren für eine Kohäsionsbetrachtung relevante Kopplungen von dieser Art der Vererbung beeinflußt werden.

Für die Kohäsionsbetrachtung folgt daraus, daß die Sicht auf die Klasse $\mathcal{B}$ zuvor entsprechend der Vererbungsbeziehung modifiziert werden muß. An die Stelle der Betrachtung der neu implementierten Funktionalität tritt die Betrachtung der tatsächlich durch $\mathcal{B}$ verfügbaren Funktionalität. Dieser Vorgang der Modifikation der Sicht auf eine Klasse entsprechend des Vererbungskontextes wird als *Flatten* bezeichnet [SiBe00].

Die Sicht auf Klasse $\mathcal{B}$ stellt sich nach dem Flatten wie in Abbildung 39 abgebildet dar. Wesentliche Schritte dieses Flatten innerhalb dieser Verwendungsart von Vererbung sind:

- Herunterkopieren aller Methoden entsprechend der Regeln zur Sichtbarkeit,
- Herunterkopieren aller Attribute entsprechend der Regeln zur Sichtbarkeit,
- Übertragung der Interaktionskopplungen von und zu Methoden der Oberklasse auf die entsprechenden Methoden der Unterklasse. Auch hier werden jeweils nur Kopplungen zu denjenigen Methoden betrachtet, die entsprechend der Regeln zur Sichtbarkeit in $\mathcal{B}$ durch ein Erben aus $\mathcal{A}$ sichtbar sind.

Die Auswirkungen einer solchen Vererbung auf klassische, knotenbasierte Größen- und Kopplungsmaße können bereits sehr deutlich am Beispiel aus Abbildung 38 und Abbildung 39 erkannt werden: $\mathcal{B}_{\text{Flatten}}$ besitzt z.B. mehr Methoden und Attribute als $\mathcal{B}$ und besitzt damit – je nach Verhalten der einzelnen, geerbten Methoden – zusätzliche Kopplungen. Die Auswirkungen des Flatten-Konzepts auf klassische, knotenbasierte Maße im Kontext realer, im Rahmen dieser Arbeit durchgeführten Projekte sind detailliert in [SiBeLe01] beschrieben.

8.2.2 Schnittstellenabstraktion

Typisches Indiz der Schnittstellenabstraktion ist die Angabe von Methodendeklarationen ohne dazugehörige Implementierungen. Während von solchen Klassen keine Instanzen gebildet werden können, dürfen sehr wohl Variablen/Attribute von diesem Typ gebildet werden, denen später Objekte von Unterklassen zugewiesen werden, in denen die geforderten Methodendeklarationen implementiert werden. Das in Abbildung 40 dargestellte Klassendiagramm in UML-Notation (vgl. [FoSc98]) soll eine typische solche Vererbungsbeziehung darstellen: Da Klasse $\mathcal{A}$ drei *rein virtuelle* Methoden (*pure virtual*, kursive Schrift innerhalb der UML) besitzt, ist es zwar nicht möglich, Instanzen dieser Klasse zu bilden, aber es ist erlaubt, Variablen/Attribute vom Typ $\mathcal{A}$ zu deklarieren. Da $\mathcal{B}$ alle drei Methodendeklarationen implementiert, können diesen Variablen/ Attributen anschließend Instanziierungen der Klasse $\mathcal{B}$ zugewiesen werden. Werden bei der Betrachtung von auf Klassenelementen basierender Klassenkohäsion (oder aller darüber liegenden Hierarchiestufen) die Interaktionskopplungen der einzelnen Methoden als Merkmale für die Zusammengehörigkeit verwendet, kann die bloße Betrachtung der auf Elemente von $\mathcal{B}$ weisenden Benutzungen zu unvollständigen Ergebnissen führen: Ein wesentlicher Vorteil der Schnittstellenabstraktion, bei der im Gegensatz zum Beispiel in Abbildung 40 mehrere Klassen dieselben Methodendeklarationen implementieren, ist die Möglichkeit der abstrakten Programmierung, in der innerhalb des Quelltextes ausschließlich Benutzungen zur Schnittstelle notiert werden, die dann zur Laufzeit von einem konkreten Objekt der Schnittstellenabstraktion ausgeführt werden. Dieses Prinzip, mittels einer Nachricht im Quelltext zur Laufzeit unterschiedliche Methodenimplementierungen ausführen zu lassen, wird als *Polymorphie* bezeichnet (vgl. z.B. [Meye97]). Daraus folgt, daß die Methoden der Implementierungsklassen innerhalb des Quelltextes niemals direkt verwendet werden.

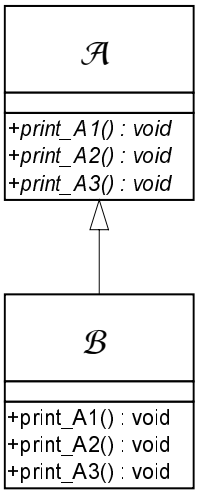


Abbildung 40: Beispiel einer Schnittstellen-abstraktion

Dieselbe Problematik tritt bei der Betrachtung von auf Klassen basierender Dateikohäsion (oder aller darüber liegenden Hierarchiestufen) auf, die Komponentenkopplungen berücksichtigen soll. Auch hier existiert i.d.R. ausschließlich eine Komponentenkopplung zu der Schnittstellenklasse; die Unterklassen treten innerhalb des Quelltextes nicht auf. Das Phänomen fehlender Kopplung läßt sich auf alle Hierarchiestufen übertragen.

Um bei einer Kohäsionsbestimmung, die auf Kopplung basieren soll, die zur Laufzeit tatsächlich existierenden Kopplungen zu Unterklassen mit betrachten zu können, muß im Rahmen des Flatten folgende Maßnahme vor einer Kohäsionsbestimmung getroffen werden:

- Alle Klassen bzw. deren Methoden, die Interaktionskopplungen zu Methoden der Schnittstellenklassen besitzen, müssen ebenfalls Interaktionskopplungen zu allen Implementierungen der entsprechenden Methoden aufweisen.

Dieser Aktivität liegt die Annahme zugrunde, daß zur Laufzeit tatsächlich alle konkreten Implementierungsklassen verwendet werden. Welche Implementierungsklassen das zur Laufzeit tatsächlich sein können, kann i.d.R. keine statische Analyse mehr aufzeigen (da die konkreten Objekte z.B. von konkreten Anwendungsfällen abhängen können). Solche zusätzlichen Kopplungen werden daher als *potentielle Interaktionskopplungen* bezeichnet [SiBe00].
Die Auswirkungen dieser Vererbungsart auf klassische, knotenbasierte Softwaremaße beschränken sich primär auf Kopplungsmaße und sind wiederum detailliert in [SiBeLe01] beschrieben

8.2.3 Typhierarchieimplementierung

Typisches Indiz der Typhierarchieimplementierung ist das sogenannte *Überschreiben* (*Overriding*) in Unterklassen von in Oberklassen deklarierten und implementierten Methoden, d.h. im Gegensatz zur Schnittstellenabstraktion sind sowohl von der Ober- als auch von der Unterklasse Instanzen zu bilden. Des weiteren wird i.d.R. innerhalb der Typhierarchieimplementierung Funktionalität von der Oberklasse in die Unterklasse vererbt. Das in Abbildung 41 dargestellte Klassendiagramm in UML-Notation (vgl. [FoSc98]) soll eine typische solche Vererbungsbeziehung darstellen. Wie bei der Schnittstellenabstraktion wird innerhalb des Quelltextes i.d.R. mit Variablen/Attributen des Typs $\mathcal{A}$ gearbeitet. Zur Laufzeit kann dies auch tatsächlich zur Benutzung von Instanziierungen der Klasse $\mathcal{A}$ führen. Es ist allerdings ebenfalls möglich, daß zur Laufzeit Objekte aus $\mathcal{B}$ die Nachrichten empfangen. Überschreibt $\mathcal{B}$ die aufgerufene Methode, wird nicht die Implementierung der Methode aus $\mathcal{A}$ verwendet, sondern die neue Implementierung aus $\mathcal{B}$ (in C++ muß dafür die entsprechende Methode in der Oberklasse zusätzlich explizit als `virtual` deklariert sein).

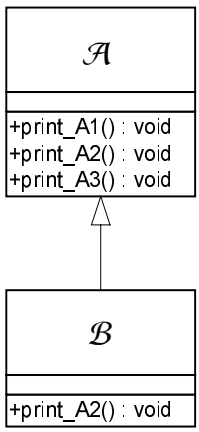


Abbildung 41: Beispiel einer Typhierarchie-implementierung

Die Probleme, die sich für die Kohäsionsbestimmung einer derart betrachteten Klasse ergeben, entsprechen denen der Schnittstellenabstraktion und der Generalisierungs/ Spezialisierungs-Vererbung:

- $\mathcal{B}$ besitzt mehr Funktionalität, als dies dem Quelltext entnehmbar ist und
- in $\mathcal{B}$ überschriebene Methoden besitzen i.d.R. keine eingehende Interaktionskopplung.

Um diese Probleme für eine Kohäsionsbetrachtung zu lösen, sind die in dem Abschnitt der Generalisierung/Spezialisierung und der Schnittstellenabstraktion vorgestellten Aktivitäten im Rahmen des Flatten notwendig.
Diese Vererbungsart hat bzgl. klassischer, knotenbasierter Softwaremaße sowohl Auswirkungen auf Größen-, als auch auf Kopplungsmaße: So besitzt eine von $\mathcal{B}$ aus Abbildung 41 gebildete Klasse $\mathcal{B}_{\text{Flatten}}$ sowohl mehr Methoden, als auch – je nach Verhalten der einzelnen geerbten Methoden – zusätzliche Kopplungen. Derartige Auswirkungen für praxisrelevante Systemgrößen sind wiederum in [SiBeLe01] beschrieben.

8.2.4 Quelltextwiederverwendung

Auch wenn die Quelltextwiederverwendung keine Motivation für Vererbung innerhalb klassischer Analyse- und Entwurfsmethoden ist, so wird sie dennoch in größeren Projekten gelegentlich eingesetzt. Indiz einer solchen Vererbungsverwendung ist, daß die Unterklasse eben nicht in einer „*ist-ein*"-Relation mit der Oberklasse steht; statt dessen steht in einem solchen Fall die Unterklasse i.d.R. in einer „*benutzt*"- oder „*hat*"-Relation zur Oberklasse. Da Objekte der Unterklasse in diesem Fall semantisch nicht Objekte der Oberklasse sind, werden die Unterklassen auch keine Zusammengehörigkeitsindizien aus der Oberklasse erben, d.h. es müssen keinerlei Aktivitäten vor einer Kohäsionsbestimmung durchgeführt werden.
Da eine durch Quelltextwiederverwendung motivierte Vererbung allerdings i.A. eine schlechte Entwurfsqualität darstellt, kann es hilfreich sein, die in den letzten drei Abschnitten vorgestellten Aktivitäten dennoch durchzuführen, d.h. die eigentliche Semantik der Vererbung – die Realisierung einer „ist-ein"-Relation – der Quelltextwiederverwendung aufzuzwingen: Eine Kohäsionsbestimmung derartiger Klassen wird i.d.R. eine geringe Kohäsion aufweisen (z.B. separierbare, abstrakte Kohäsion, vgl. Kapitel 7.1.2), da die Unterklasse mit der Oberklasse bis auf deren teilweise Verwendung i.d.R. wenig zusammengehört: Statt einer durch das Flatten entstandenen kohäsiven Klasse wird unter der Voraussetzung, daß sowohl Ober- als auch Unterklasse kohäsiv sind, eine nicht kohäsive Klasse mit zwei deutlich kohäsiven Teilen entstehen, die den ursprünglichen Klassen entsprechen. Die Anwendung der distanzmaßbasierten Kohäsionsbestimmung für das Refactoring (vgl. Kapitel 9.3.2) kann u.a. diese Art von Indizien zur Entdeckung einer ungeeigneten Vererbungsbeziehung entdecken (vgl. Abschnitt 8.4).

8.3 Klassenkopplungbasierte Kohäsion

In diesem Abschnitt werden einige Beispiele einer auf Klassenkopplung basierenden Kohäsionsbetrachtung vorgestellt, d.h. innerhalb des entsprechenden Prozesses aus Kapitel 7.3.1 ist das Ergebnis des Schritts 2 — die Bestimmung derjenigen Hierarchiestufe, deren Elemente für die Kohäsion der darüber liegenden Stufen relevant sind — auf Klassen festgelegt.
Mögliche Ergebnisse des innerhalb des Prozesses geforderten nächsten Schritts — die Festlegung der Kopplungsart, die zwischen den Elementen betrachtet werden soll — können den bekannten Kopplungsarten entnommen werden (vgl. Kapitel 7.2): Vererbungskopplung, Komponentenkopplung und Interaktionskopplung. Da die letzten beiden Kopplungsarten eng miteinander verknüpft sind (vgl. Kapitel 7.2.2), werden sie im folgenden, nach der Beschäftigung mit der Vererbungskopplung, gemeinsam betrachtet.

8.3.1 Vererbungbasierte Kohäsion

Innerhalb der vererbungbasierten Kopplung zwischen Klassen sind zwei Klassen genau dann gekoppelt, wenn sie in einer direkten oder indirekten Vererbungsbeziehung stehen. Als Vererbungsbeziehungen werden Generalisierung/Spezialisierung, Schnittstellenabstraktion, und Typhierarchieimplementierung betrachtet (vgl. Abschnitt 8.2). Die Quelltextwiederverwendung wird als Kombination aller anderen drei Vererbungsformen betrachtet und daher im folgenden nicht weiter behandelt.
Die typische Darstellung einer Menge von Klassen und deren Vererbungsrelationen geschieht mittels Klassendiagrammen, in denen jegliche Formen von Assoziationen ignoriert werden. In Abbildung 42 ist ein Beispiel eines solchen Klassendiagramms in UML-Notation dargestellt.

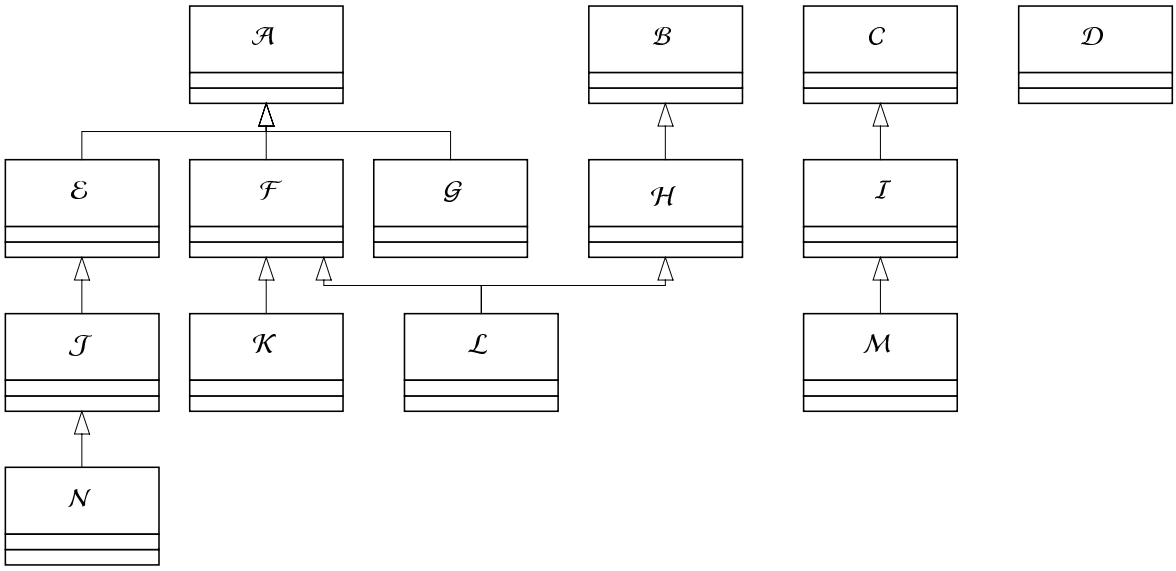


Abbildung 42: Beispiel eines Assoziationen ignorierenden Klassendiagramms

Für die Betrachtung dieser Kopplungen sind zwei verschiedene Ansätze denkbar:

- *Ungewichtete Vererbung*, d.h. zwei Klassen stehen (oder stehen nicht) in einer Vererbungsbeziehung. Die verschiedenen Arten der Vererbung und deren Auswirkungen auf die Kohäsion (vgl. Kapitel 8.2. und 7.2.3) werden nicht unterschieden. Diese Betrachtungsweise entspricht der Darstellung innerhalb eines Klassendiagramms.
- *Gewichtete Vererbung*, d.h. die Vererbungsbeziehung zwischen zwei Klassen wird gewichtet, um die verschiedenen Vererbungsarten entsprechend abzubilden.

Beide Varianten sollen im folgenden vorgestellt werden.

Ungewichtete Vererbung

Innerhalb dieser Betrachtungsweise wird lediglich berücksichtigt, ob eine Klasse Ober- oder Unterklasse einer anderen Klasse ist; die konkrete Art der verwendeten Vererbung wird für diese Sichtweise als irrelevant betrachtet. Aus dieser Subsumption folgt, daß eine Klasse nicht nur von den direkten Oberklassen, sondern auch von den indirekten Oberklassen abhängt, da eine Änderung in einer dieser Oberklassen Auswirkungen auf die Unterklasse haben kann. Dabei spielt es keine Rolle, ob von n direkten Oberklassen geerbt wird, oder ob die n Oberklassen einen langen Vererbungspfad darstellen. Für den Schritt 4 – die Identifikation der die jeweilige Kopplungsart begründenden Merkmale – folgt daraus, daß die Merkmale einer Klasse durch die Menge derjenigen Klassen gegeben sind, von denen die Klasse potentiell etwas erbt. Um der Vorstellung gerecht zu werden, daß diese Merkmale genau die Klassenmenge darstellen, auf die die Klasse ihre Funktionalität verteilt haben kann, ist es sinnvoll, die Klasse selbst ebenfalls dieser Merkmalsmenge beizufügen. Damit ergibt sich die in Definition 36 angegebene *allgemeine Vererbungskohäsion*: Sie besteht aus dem zugrundeliegenden Konstruktionsverfahren, der für die entsprechende Realisierung notwendigen Merkmalsmenge und der bzgl. dieser Merkmalsmenge anschließend vorzunehmenden Interpretation:

Konstruktion der allgemeinen Vererbungskohäsion: Mehrere Klassen gehören entsprechend der allgemeinen Vererbungskohäsion um so mehr zusammen, um so ähnlicher die Mengen ihrer Oberklassen sind:

Merkmalsmenge der allgemeinen Vererbungskohäsion: Für eine Klasse C ist die Menge der Merkmale gegeben durch:
$\mathcal{M}$ = {C} ∪ {direkte Oberklassen von C} ∪ {indirekte Oberklassen von C}.

Interpretation der allgemeinen Vererbungskohäsion: Klassen mit einer engen Distanz zueinander besitzen untereinander Vererbungskopplung (im Falle einer direkten Vererbungsbeziehung) oder besitzen ähnliche Vererbungskopplung zu gemeinsamen Oberklassen (im Falle keiner direkten Vererbungsbeziehung). Daher verhalten sich diese Klassen vermutlich nach außen ähnlich, stellen also vermutlich semantisch ähnliche Konzepte dar.

Definition 36: Allgemeine Vererbungskohäsion

Die vorsichtige Interpretationsform ist notwendig, da die verschiedenen Vererbungsformen undifferenziert und ungewichtet betrachtet werden. Große semantische Unterschiede zwischen Ober- und Unterklasse, was sich z.B. in einem sehr schmalen Interface und einer neben dieser Schnittstelle viel zusätzliche Funktionalität implementierenden Klasse äußern kann, werden nicht erkannt.
Diese Form der Kohäsionsbetrachtung ergibt für das Beispiel aus Abbildung 42 für die einzelnen Klassen folgende Merkmalsmengen:

$\mathcal{A}$: {$\mathcal{A}$}	$\mathcal{E}$: {$\mathcal{E}$, $\mathcal{A}$}	$\mathcal{I}$: {$\mathcal{I}$, $\mathcal{C}$}	$\mathcal{L}$: {$\mathcal{L}$, $\mathcal{F}$, $\mathcal{H}$, $\mathcal{A}$ $\mathcal{B}$}
$\mathcal{B}$: {$\mathcal{B}$}	$\mathcal{F}$: {$\mathcal{F}$, $\mathcal{A}$}	$\mathcal{J}$: {$\mathcal{J}$, $\mathcal{E}$, $\mathcal{A}$}	$\mathcal{M}$: {$\mathcal{M}$, $\mathcal{I}$, $\mathcal{C}$}
$\mathcal{C}$: {$\mathcal{C}$}	$\mathcal{G}$: {$\mathcal{G}$, $\mathcal{A}$}	$\mathcal{K}$: {$\mathcal{K}$, $\mathcal{F}$, $\mathcal{A}$}	$\mathcal{N}$: {$\mathcal{N}$, $\mathcal{J}$, $\mathcal{E}$, $\mathcal{A}$}
$\mathcal{D}$: {$\mathcal{D}$}	$\mathcal{H}$: {$\mathcal{H}$, $\mathcal{B}$}		

Die daraus errechenbare Distanzmatrix (Schritt 5) läßt sich z.B. mittels des Spring-Embedders in eine dreidimensionale Darstellung überführen, die dann anschließend z.B. mittels eines VRML-Clients explorierbar ist (vgl. auch Kapitel 9.1) [25].
Eine typische, derart generierte Sichtweise auf das System aus Abbildung 42 ist in Abbildung 43 dargestellt: Jede Klasse wird dabei durch eine gefärbte Kugel gleicher Größe innerhalb eines dreidimensionalen Raums dargestellt. Die Tiefendimension, die für die Entfernung zwischen Klassen gleichwertig zur horizontalen und vertikalen Dimension verwendet wird, wird durch Verkleinern/Vergrößern der Objekte in Entsprechung der Tiefe dargestellt, d.h. weiter in der Tiefe liegende Objekte sind klein dargestellt.

[25] Die Vorteile einer derartigen, rechnergestützten Darstellung, die u.a. in den vielfältigen Interaktionsmöglichkeiten liegen, die der Betrachter innerhalb dieser virtuellen Welten besitzt (z.B. die Möglichkeit, mittels des Zoomens das Betrachten auf Teile der Visualisierung zu fokussieren („*regions of interest*"), vgl. z.B. [DäPa98]), gehen in einem zweidimensionalen Screenshot verloren: Dies betrifft nicht nur die fehlende Interaktion, sondern ebenfalls die Tiefeninformation, die innerhalb des VRML-Clients durch Farbschattierungen, Bewegen in der dargestellten Welt oder durch die Verwendung spezieller Hardware (z.B. *Shutter Brillen*), vermittelt wird. Um dennoch innerhalb einer gedruckten Arbeit einen ungefähren Eindruck der Technik der dreidimensionalen Softwarevisualisierung zu vermitteln, kann im folgenden auf Screenshots nicht verzichtet werden. Um zumindest einen Eindruck von den verwendeten Farben, die ebenfalls räumliche Informationen darstellen, zu erhalten, sind im folgenden alle mit einem (*) versehenen Abbildungen im Anhang noch einmal als Farbausdruck auf Seitengröße beigefügt.

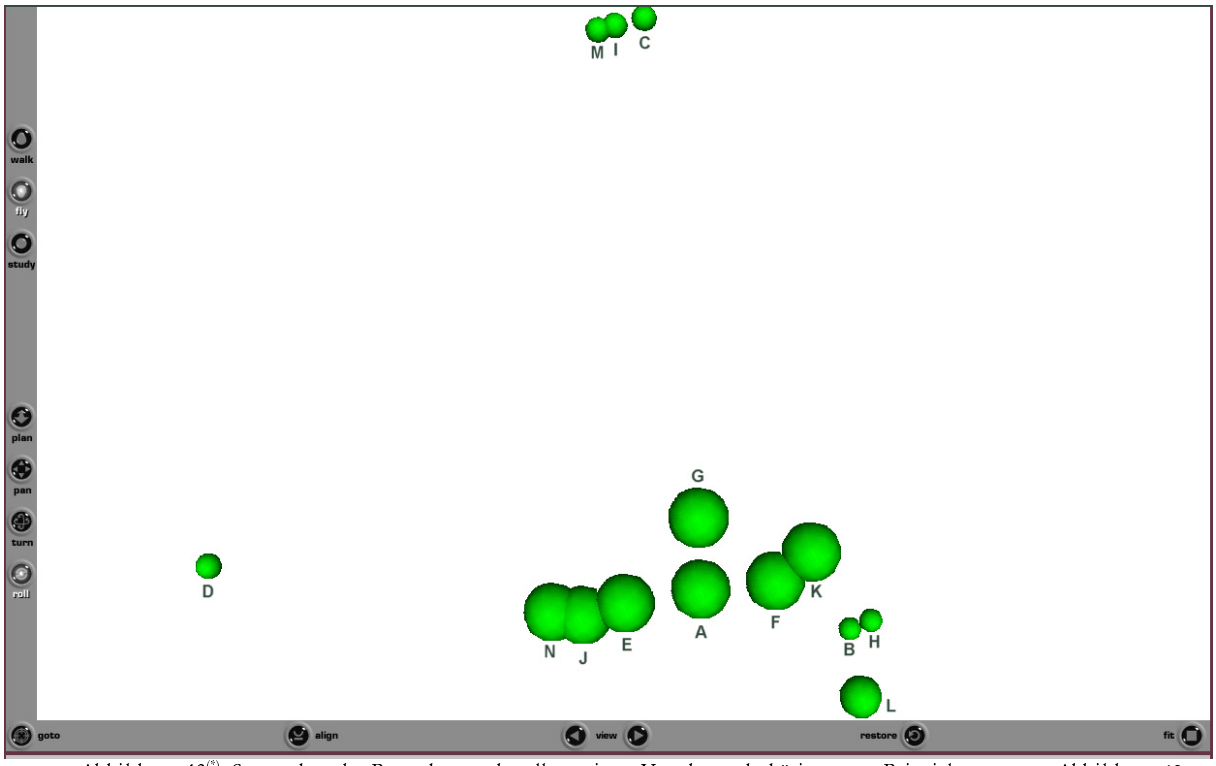


Abbildung 43(*): Screenshot der Betrachtung der allgemeinen Vererbungskohäsion vom Beispielsystem aus Abbildung 42

Deutlich sind die folgenden Aussagen der Visualisierung, die vom Quelltextparsen bis zur Erstellung der dreidimensionalen Welt vollständig automatisiert abläuft, abzuleiten:

- Es gibt drei Gruppen von Klassen, in denen die Klassen jeweils eine enge Vererbungskopplung besitzen (G_1={$\mathcal{M}$, $\mathcal{I}$, $\mathcal{C}$}, G_2={$\mathcal{B}$, $\mathcal{H}$} und G_3={$\mathcal{N}$, $\mathcal{J}$, $\mathcal{E}$, $\mathcal{G}$, $\mathcal{A}$, $\mathcal{F}$, $\mathcal{K}$}).
- Die Klasse $\mathcal{L}$ liegt zwischen den Gruppen G_2 und G_3 (im zweidimensionalen Screenshot lediglich an der Darstellungsgröße erkennbar, die entsprechend der Tiefeninformation zwischen den Darstellungsgrößen der Elemente aus G_2 und G_3 liegt) und besitzt folglich Vererbungskopplungen zu beiden Klassengruppen.
- $\mathcal{D}$ besitzt keinerlei Vererbungskopplungen zu Klassen innerhalb des betrachteten Systems.
- Innerhalb der Gruppe G_3 können die Untergruppen $G_{3,1}$={$\mathcal{N}$, $\mathcal{J}$, $\mathcal{E}$}, $G_{3,2}$={$\mathcal{G}$} und $G_{3,3}$={$\mathcal{F}$, $\mathcal{K}$} identifiziert werden, die alle gleich starke Vererbungskopplungen zur zentralen Klasse $\mathcal{A}$ besitzen. Die Eigenschaft ‚zentral' wird dabei innerhalb der Visualisierung direkt umgesetzt.
- Die Gruppe G_1 liegt aufgrund der von Klassen aus G_2 und G_3 unabhängigen Vererbungsstruktur separat.

Ein wesentlicher Vorteil dieser *dreidimensionalen Vererbungsgraphen*, der bei derartigen Systemgrößen noch nicht relevant ist, ist die Skalierbarkeit bzgl. der folgenden beiden Parameter:
- Anzahl der betrachteten Klassen und
- Anzahl der betrachteten Vererbungsbeziehungen.

Speziell der letzte Parameter nimmt in typischen JAVA-Systemen eine gewichtige Rolle ein, da dort die Schnittstellenabstraktion aufgrund der Sprachunterstützung häufig eingesetzt wird. So ist z.B. in Abbildung 44 ein sehr kleiner Ausschnitt einer typischen, ebenfalls von Werkzeugen generierten Klassenhierarchie eines Systems dargestellt, das 645 Klassen und 295 Vererbungsrelationen besitzt. In ganzer Größe ist eine derartige, baumorientierte 2D-Sicht nicht zu betrachten. Die entsprechende, dreidimensionale Darstellung desselben Systems ist vollständig in Abbildung 45 dargestellt: Hierbei wurden die einzelnen Klassen zusätzlich entsprechend der Subsystemzugehörigkeit koloriert, d.h. alle Klassen eines Subsystems besitzen dieselbe Farbe, und die Größe der einzelnen Klassen proportional zur Anzahl direkter Unterklassen gewählt. Darüber hinaus wurden die direkten Vererbungsbeziehungen eingezeichnet. Die Rolle einer Klasse (Ober- bzw. Unterklasse) ist durch einen Farbverlauf der Linie dargestellt (vgl. besonders den entsprechenden Farbausdruck im Anhang).

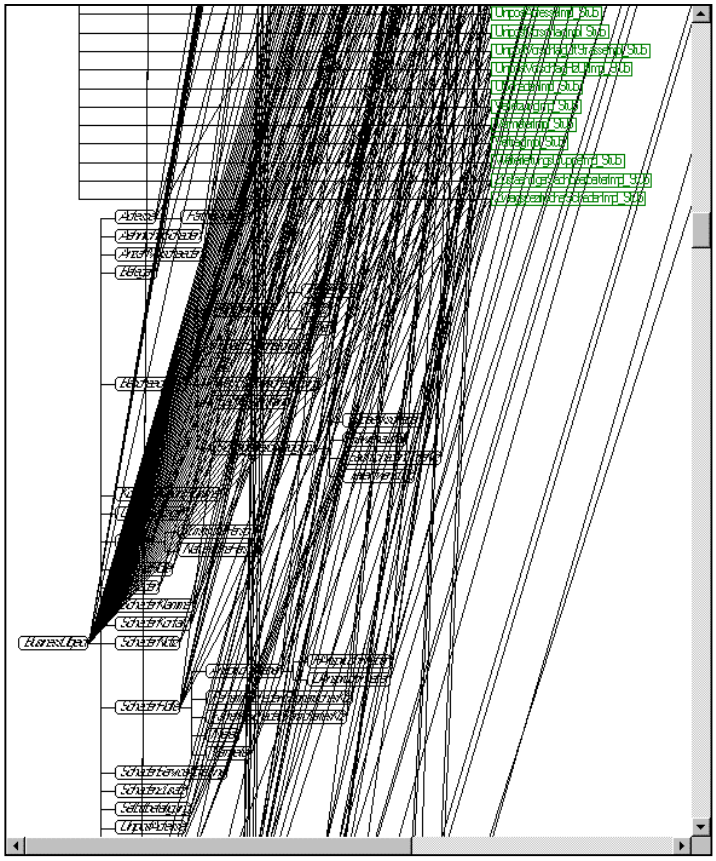


Abbildung 44: Vererbungsgraph innerhalb eines kommerziellen CASE-Tools

Bei Visualisierungen dieser Größe gehen aufgrund der Dimensionsreduktion einige wenige Informationen verloren. So sind von den innerhalb der Distanzmatrix existierenden Ordnungsrelationen nach 2 Stunden nichtlinearer Optimierung mittels des Spring-Embedders auf einem normalen PC ca. 88% erhalten, fast 12% der Ordnungsrelationen werden verletzt, d.h. dort ist die Ordnungsrelation zwischen zwei Distanzen zwischen jeweils 2 Klassen falsch dargestellt. In praktischen Anwendungen spielt dieser Informationsverlust allerdings kaum eine Rolle (vgl. Kapitel 9).

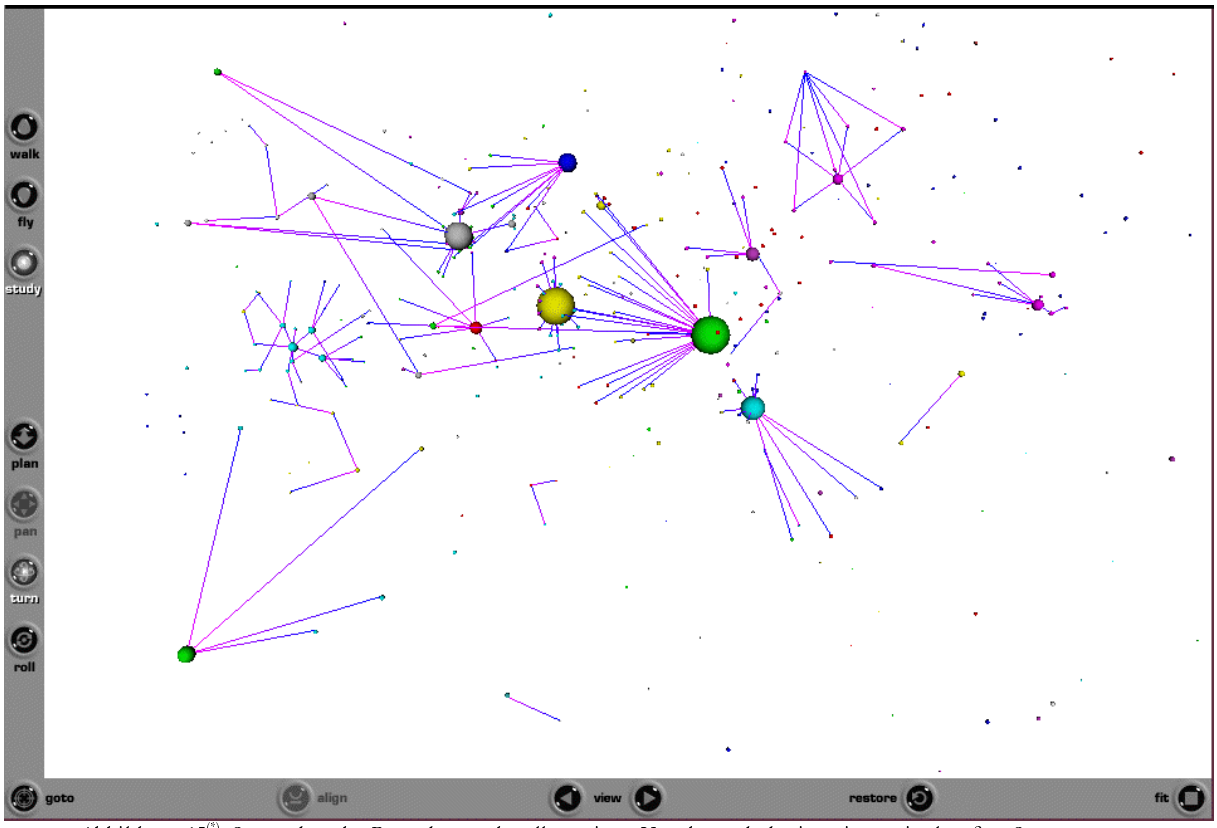


Abbildung 45(*): Screenshot der Betrachtung der allgemeinen Vererbungskohäsion eines mittelgroßen Systems

Gewichtete Vererbung

Entsprechend der unterschiedlichen Arten von Vererbung kann die Eigenschaft, mit einer Oberklasse in einer Vererbungsrelation zu stehen, durch eine Gewichtung weiter verfeinert werden. Für die einzelnen Vererbungsarten kann eine verfeinerte Kopplung wie folgt beschrieben werden:

- *Generalisierung/Spezialisierung*: Eine Unterklasse, deren Erben von einer Oberklasse eine Generalisierung/Spezialisierung darstellt, ist um so mehr mit dieser Oberklasse gekoppelt, um so mehr Funktionalität von der Oberklasse in die Unterklasse vererbt wird, d.h. um so weniger umfangreich die Spezialisierung stattfindet. Die Kohäsion zwischen zwei Klassen liegt dabei zwischen den beiden Extremfällen *maximale Generalisierungs-/Spezialisierungskopplung*, innerhalb derer die Unterklasse keine zusätzliche Funktionalität hinzufügt, d.h. keine Spezialisierung vornimmt, und der *minimalen Generalisierungs-/Spezialisierungskopplung*, innerhalb derer die Unterklasse keine Funktionalität aus der Oberklasse übernimmt (z.B. möglich durch entsprechende Sichtbarkeiten in der Oberklasse).
- *Schnittstellenabstraktion*: Eine Unterklasse, deren Erben von einer Oberklasse eine Schnittstellenabstraktion darstellt, ist um so mehr mit dieser Oberklasse gekoppelt, um so mehr Methoden der Unterklasse lediglich eine Implementierung der Schnittstelle darstellen. Die Kohäsion zwischen zwei Klassen liegt dabei zwischen den beiden Extremfällen *maximale Schnittstellenabstraktionskopplung*, innerhalb derer die Unterklasse lediglich die Schnittstelle implementiert, d.h. keine zusätzliche Funktionalität anbietet, und der *minimalen Schnittstellenabstraktionskopplung*, innerhalb derer die Unterklasse keine Methode implementiert, selbst aber Funktionalität anbietet (z.B. möglich bei leeren Schnittstellen oder Vererbung zwischen Schnittstellen).
- *Typhierarchieimplementierung*: Eine Unterklasse, deren Erben von einer Oberklasse eine Typhierarchieimplementierung darstellt, ist um so mehr mit dieser Oberklasse gekoppelt, je ähnlicher die von beiden Klassen angebotene Funktionalität ist. Die Kohäsion zwischen zwei

Klassen liegt dabei zwischen den beiden Extremfällen *maximale Typhierarchieimplementierungskopplung*, innerhalb derer die Unterklasse genau die Funktionalität anbietet, die auch die Oberklasse anbietet, und der *minimalen Typhierarchieimplementierungskopplung*, innerhalb derer die Unterklasse eine vollständig neue Funktionalität anbietet. Innerhalb der maximalen Typhierarchieimplementierungskopplung ist es irrelevant, ob eine Methode aus der Oberklasse direkt geerbt wird, oder ob die Methode in der Unterklasse neu definiert wird. Nach außen darf dieser Unterschied ebenfalls nicht ersichtlich sein, da anderenfalls eine wesentliche Eigenschaft der Typhierarchie, nämlich die Austauschbarkeit von Klassen, verletzt ist.

Unter der Funktionalität einer Klasse wird üblicherweise die Menge öffentlicher Methoden verstanden. Je nach System (und den verwendeten Kodierrichtlinien) ist es allerdings ebenfalls möglich, öffentliche Attribute als Funktionalität aufzufassen: Damit ist es möglich, die aufgrund der üblichen Verwendung von Attributen mittels eigener, öffentlicher Zugriffsmethoden verfeinerte Kopplung zwischen Ober- und Unterklasse auch aus der methodenlosen, direkten Attributsbenutzung abzuleiten.
Alle drei auf Vererbungskopplungen basierende Kohäsionsbetrachtungen lassen sich ausgehend von dieser Kopplungsverfeinerung mittels folgender Definition betrachten:

Konstruktion der gewichteten Vererbungskohäsion: Mehrere Klassen gehören entsprechend der gewichteten Vererbungskohäsion um so mehr zusammen, um so mehr gemeinsame Methoden und Attribute sie nach außen anbieten.

Merkmalsmenge der gewichteten Vererbungskohäsion: Für eine Klasse C ist die Menge der zu betrachtenden Merkmale gegeben durch:
$\mathcal{M}$ = {Öffentliche, in C neu implementierte oder deklarierte Funktionalität}
$\cup$ {von direkten Oberklassen an C geerbte implementierte oder deklarierte Funktionalität}.

Die Sicht auf die Oberklasse muß dabei deren Flatten-Version berücksichtigen, da die Oberklasse selbst ebenfalls Funktionalität erben kann.

Interpretation der gewichteten Vererbungskohäsion: Klassen mit einer geringen Distanz zueinander erben gleiche Funktionalität aus gemeinsamen Oberklassen und fügen selbst wenig neue hinzu (im Fall keiner direkten Vererbungsbeziehung) oder die Klassen stehen untereinander in einer Vererbungsbeziehung, wobei Unterklassen aber jeweils nur wenig Funktionalität hinzufügen. In beiden Fällen verhalten sich die Klassen nach außen sehr ähnlich, da die Menge der an ihre Objekte zu verschickenden Nachrichten sehr ähnlich ist.

Definition 37: Gewichtete Vererbungskohäsion

Bzgl. dieser Definition müssen für die einzelnen Vererbungsarten hinsichtlich der Konstruktion, der Merkmalsmenge und der Interpretation folgende Anmerkungen gemacht werden:
Für eine Oberklasse $\mathcal{A}$ und einer Unterklasse $\mathcal{B}$ gilt bzgl.

- der Generalisierungs-/Spezialisierungskohäsion:
  - Beide Klassen $\mathcal{A}$ und $\mathcal{B}$ müssen in der Flatten-Version vorliegen, da anderenfalls z.B. die Funktionalität in einer Unterklasse, die keine zusätzliche Funktionalität implementiert, leer wäre, d.h. maximale Generalisierungs-/Spezialisierungskohäsion suggeriert wird, die Unterklasse aber durch eine zusätzliche Oberklasse $\mathcal{A}_2$ sehr wohl eine starke Spezialisierung vornehmen kann.
  - Erbt $\mathcal{B}$ ausschließlich von $\mathcal{A}$ und implementiert oder deklariert $\mathcal{B}$ keine zusätzliche Funktionalität, so sind die Merkmalsmengen beider Klassen identisch, d.h. ihre Kohäsion maximal (schließlich sind beide Klassen nach außen identisch).

  - Je mehr Funktionalität die Unterklasse zusätzlich anbietet (z.B. durch eine zusätzliche Oberklasse von $\mathcal{B}$) desto unterschiedlicher sind die Merkmalsmengen beider Klassen, d.h. desto weniger kohäsiv sind $\mathcal{A}$ und $\mathcal{B}$ (schließlich stellt $\mathcal{B}$ zunehmend eine Spezialisierung dar; je mehr spezialisiert wird, desto weniger relevant ist die Kopplung zu $\mathcal{A}$).
- der Schnittstellenabstraktionskohäsion:
  - Beide Klassen $\mathcal{A}$ und $\mathcal{B}$ müssen in der Flatten-Version vorliegen, da das zu implementierende Interface ($\mathcal{A}$) selbst weitere Oberklassen besitzen kann, d.h. Signaturen von Funktionalität erbt, die ebenfalls in der $\mathcal{A}$ implementierenden Klasse ($\mathcal{B}$) implementiert werden müssen.
  - Implementiert $\mathcal{B}$ ausschließlich die durch $\mathcal{A}$ deklarierte Schnittstelle, besitzt keine weitere Oberklasse und deklariert oder implementiert keine zusätzlichen Funktionen, so sind die Merkmalsmengen beider Klassen identisch, d.h. ihre Kohäsion maximal (schließlich können der Unterklasse ausschließlich Nachrichten geschickt werden, die in der Oberklasse definiert sind).
  - Je mehr Funktionalität $\mathcal{B}$ zusätzlich anbietet (z.B. durch eine zusätzliche Oberklasse), desto unterschiedlicher sind die Merkmalsmengen beider Klassen, d.h. desto weniger kohäsiv sind $\mathcal{A}$ und $\mathcal{B}$ (schließlich stellt $\mathcal{B}$ weit mehr Funktionalität zur Verfügung, als die Schnittstelle $\mathcal{A}$ vorgibt).
- der Typhierarchieimplementierungskohäsion:
  - Beide Klassen $\mathcal{A}$ und $\mathcal{B}$ müssen in der Flatten-Version vorliegen, da anderenfalls Unterklassen, die keine Methoden der Oberklasse überschreiben, eine leere Merkmalsmenge besitzen, sich aber eigentlich zur Oberklasse identisch verhalten und damit von ihr vollständig abhängen.
  - Implementiert oder deklariert $\mathcal{B}$ keine weitere Funktionalität und besitzt $\mathcal{B}$ auch keine weiteren Oberklassen, so sind die Merkmalsmengen beider Klassen identisch, d.h. ihre Kohäsion maximal (schließlich ist $\mathcal{B}$ vollständig von $\mathcal{A}$ abhängig).
  - Je mehr Funktionalität $\mathcal{B}$ zusätzlich anbietet (z.B. durch eine weitere Oberklasse) desto unterschiedlicher sind die Merkmalsmengen beider Klassen, d.h. desto weniger kohäsiv sind $\mathcal{A}$ und $\mathcal{B}$ (schließlich ist $\mathcal{B}$ nicht mehr vollständig von $\mathcal{A}$ abhängig).

Für Quelltexte, deren verwendete Programmiersprachen Sprachkonstrukte anbieten, die eine eindeutige Zuordnung einer Vererbung zu einer der vier Verwendungen ermöglicht, können darüber hinaus zusätzliche Kohäsionsbetrachtungen angestellt werden. So kann es in solchen Fällen z.B. sinnvoll sein, lediglich Schnittstellenimplementierungen zu berücksichtigen, um die Abhängigkeiten der Schnittstellen und der sie jeweils implementierenden Klassen zu betrachten. Für die im objektorientierten Bereich verbreitetsten Programmiersprachen C++ und JAVA ist dies allerdings nicht möglich, da C++ keinerlei Sprachkonstrukt zur expliziten Schnittstellenabstraktion bereit stellt, d.h. alle vier Vererbungsarten mittels einer einheitlichen Syntax darstellt, und JAVA's Schnittstellenabstraktion (mittels des Sprachelements `implements`) nicht konsequent genug umgesetzt wurde: so ist es möglich, Attribute in Schnittstellen zu implementieren, die an die implementierenden Klassen weiter vererbt werden, d.h. die Reinform der Schnittstellenvererbung kann dadurch mit einer Generalisierungs-/Spezialisierungsvererbung gemischt werden.

8.3.2 Interaktionbasierte Kohäsion

Innerhalb der interaktionbasierten Kopplung zwischen Klassen sind zwei Klassen genau dann gekoppelt, wenn die eine direkt etwas von der anderen verwendet. Unter einer direkten Verwendung der Klasse $\mathcal{B}$ von einer Klasse $\mathcal{A}$ wird dabei verstanden, wenn mindestens ein Element aus $\mathcal{A}$ mindestens eine Interaktionskopplung (vgl. Kapitel 7.2.1) zu mindestens einem Element aus $\mathcal{B}$ besitzt. Entsprechend der üblichen Kommunikationskanäle innerhalb objektorientierter Programmierung kann eine Interaktionskopplung zwischen Klassenelementen unterschiedlicher Klassen durch folgende Konzepte realisiert werden:

- Eine Methode aus $\mathcal{A}$ ruft eine Methode aus $\mathcal{B}$ auf. Diese Form der Kopplung kann in Anlehnung an das in Kapitel 7.2.1 dargestellte Kopplungsspektrum als datenbasierte Kopplung zwischen den Klassen $\mathcal{A}$ und $\mathcal{B}$ aufgefaßt werden, da auf diesem Wege lediglich wohlstrukturierte, nicht-globale Datenelemente in Form einfacher (z.B. Basistypen) oder homogener (z.B. Array) Parameter ausgetauscht werden können. Besitzt der übergebene Parameter eine eigene, nicht-triviale Struktur (z.B. in Form eines `structs` in C++) kann diese Interaktionsart auch als strukturelle Kopplung aufgefaßt werden.
- Eine Methode aus $\mathcal{A}$ verwendet ein Attribut aus $\mathcal{B}$. Diese Form der Kopplung kann in Anlehnung an das Kopplungsspektrum als externe Kopplung zwischen den Klassen aufgefaßt werden, da ein derart zu verwendendes Attribut in objektorientierten Systemen explizit für solche Interaktionen deklariert werden muß[26] (in C++ z.B. durch Setzen der Sichtbarkeit auf `public`).

Die übliche Darstellung einer Menge von Klassen und deren Interaktionskopplungen geschieht mittels Klassendiagrammen, wie z.B. innerhalb der UML vorgeschlagen (vgl. [FoSc98]). Während innerhalb des Klassendiagramms in UML eine Klasse $\mathcal{A}$ drei verschiedene Beziehungen zu einer Klasse $\mathcal{B}$ besitzen kann (Assoziation, Aggregation und Komposition), um den Grad der Verantwortlichkeit der Klasse $\mathcal{A}$ für die Klasse $\mathcal{B}$ modellieren zu können, wird dies innerhalb der statischen Produktmodellierung nicht unterschieden, da einerseits das Abbilden der drei unterschiedlichen Arten auf eine konkrete Programmiersprache nicht einheitlich geschieht (vgl. z.B. verschiedene Lösungswege für C++ in [Booc94], S. 140ff) und andererseits eine statische Quelltextanalyse nicht sicher feststellen kann, wer zur Laufzeit Objekte für wen erstellt und wieder löscht; letztere Feststellung ist z.B. wichtig für die Unterscheidung zwischen Aggregation und Komposition, hängt z.T. aber wesentlich vom jeweilig verwendeten Kontrollfluß ab, der wiederum durch unterschiedliche Anwendungsfälle des Systems bestimmt sein kann.
In der folgenden Abbildung ist ein Beispiel eines solchen die Assoziationsbeziehungen nicht weiter verfeinernden Klassendiagramms dargestellt. Innerhalb der UML wird dabei die einfache Interaktionskopplung von $\mathcal{A}$ nach $\mathcal{B}$ als gerichteter Pfeil von $\mathcal{A}$ nach $\mathcal{B}$ dargestellt; benutzen sich $\mathcal{A}$ und $\mathcal{B}$ wechselseitig, so wird eine einfache Linie eingefügt.

[26] Die Möglichkeit, speziell in hybriden Sprachen wie C++ solche explizit zu deklarierenden Interaktionen zu verbergen (z.B. mittels des `Friend`-Operators, vgl. [Schi98]), soll hier nicht weiter behandelt werden, da bei objektorientiertem Vorgehen derartige Konstrukte nicht identifiziert werden.

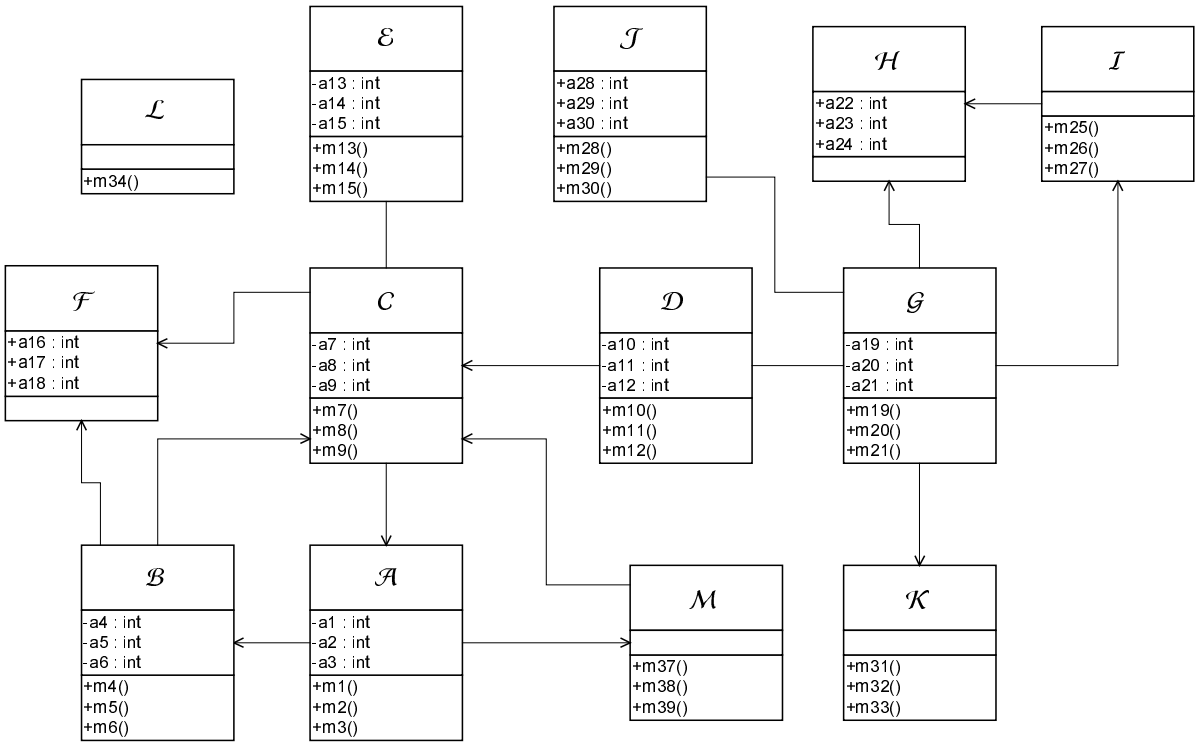


Abbildung 46: Beispiel eines Assoziationen nicht verfeinernden Klassendiagramms

Es ist bei der Interaktionskopplung wichtig, die besondere Rolle der Vererbung zu berücksichtigen: Da Kopplungen nicht nur zu Oberklassen, sondern ebenfalls zu Unterklassen existieren (entweder durch Polymorphie oder geerbte Methoden) und Benutzungen zu anderen Klassen an Unterklassen weiter vererbt werden können, sind alle Klassen vor einer Kohäsionsbetrachtung entsprechend des in Abschnitt 8.1 vorgestellten Flatten-Konzepts zu bearbeiten. Die Vererbungsrelationen selbst sind anschließend in das Klassendiagramm „eingewebt", brauchen folglich auch nicht mehr separat betrachtet zu werden.

Für die Betrachtung dieser Interaktionskopplungen sind – wie bei der Vererbungskopplung – zwei verschiedene Ansätze denkbar:

- *Ungewichtete Interaktion*, d.h. zwei Klassen stehen (oder stehen nicht) bzgl. der Interaktionskopplung in einer Beziehung. Die beiden Arten der Kopplung – datenbasierte/strukturelle und externe Kopplung – werden nicht weiter unterschieden. Diese Betrachtungsweise entspricht der Darstellung innerhalb des Klassendiagramms, da das Einfügen einer Assoziation keine Aussagen darüber macht, ob eine oder mehrere Methoden oder Attribute verwendet werden (vgl. Abbildung 46).
- *Gewichtete Interaktion*, d.h. die Interaktionskopplung zwischen zwei Klassen wird gewichtet, um die verschiedenen Kopplungsarten entsprechend abzubilden.

Beide Varianten sollen im folgenden vorgestellt werden.

Ungewichtete Interaktion

Innerhalb dieser Betrachtungsweise wird lediglich berücksichtigt, ob zwischen zwei Klassen überhaupt eine Interaktionskopplung besteht; die konkrete Art der verwendeten Interaktion ist irrelevant. Für den innerhalb des Prozesses zur kopplungbasierten Kohäsionsbetrachtung geforderten Schritt 4 – die Identifikation der die jeweilige Kopplungsart begründenden Merkmale,

vgl. Kapitel 7.3.1 – folgt daraus, daß die Merkmale einer Klasse C durch die Menge derjenigen Klassen gegeben sind, von denen C etwas verwendet. Um der Vorstellung gerecht zu werden, daß diese Merkmale genau die Klassenmenge darstellt, von denen eine Klasse Methoden oder Attribute verwendet, ist es sinnvoll, die Klasse selbst ebenfalls dieser Merkmalsmenge beizufügen. Damit ergibt sich die in Definition 38 angegebene *ungewichtete Interaktionskohäsion*:

Konstruktion der ungewichteten Interaktionskohäsion: Mehrere Klassen gehören entsprechend der ungewichteten Interaktionskohäsion um so mehr zusammen, um so mehr gemeinsame Klassen (inkl. ihrer selbst) sie jeweils verwenden.

Merkmalsmenge der ungewichteten Interaktionskohäsion: Für eine Klasse C ist die Menge der zu betrachtenden Merkmale gegeben durch:
$\mathcal{M}$ = {C} ∪ {Klassen, zu denen C mindestens eine ausgehende, direkte Interaktionskopplung besitzt}.
Die Sicht auf C muß dabei deren Flatten-Version berücksichtigen, da C Interaktionskopplungen erben und aufgrund polymorpher Aufrufe zusätzliche, potentielle Interaktionskopplungen zu Unterklassen bekommen kann.

Interpretation der ungewichteten Interaktionskohäsion: Klassen mit einer geringen Distanz zueinander besitzen eine ähnliche Menge von Klassen, von denen sie etwas verwenden. Die Ähnlichkeit schließt eine Interaktion zwischen den Klassen mit ein. Aufgrund der ähnlichen Kontexte, mit denen dicht beieinander liegende Klassen arbeiten, realisieren die betrachteten Klassen vermutlich eine ähnliche Funktionalität (im Fall keiner direkten Interaktionskopplungen zwischen den Klassen) oder implementieren in Form von auf Klassen aufgeteilten Teilfunktionalitäten vermutlich eine größere, auf höherem Abstraktionsniveau liegende Funktionalität (im Fall vieler direkter Interaktionskopplungen zwischen den Klassen).

Definition 38: Ungewichtete Interaktionskohäsion

Die vorsichtige Interpretationsform ist notwendig, da die verschiedenen Interaktionskopplungen undifferenziert und ungewichtet betrachtet werden. So wird z.B. nicht zwischen einem einzigen Methodenaufruf zu einer Klasse und dem Verwenden der kompletten, von dieser Klasse angebotenen Funktionalität unterschieden.
Diese Form der Kohäsionsbetrachtung ergibt für das Beispiel aus Abbildung 46 für die einzelnen Klassen folgende Merkmalsmengen:

$\mathcal{A}$: {$\mathcal{A}$, $\mathcal{B}$, $\mathcal{M}$}	$\mathcal{E}$: {$\mathcal{E}$, $\mathcal{C}$}	$\mathcal{H}$: {$\mathcal{H}$}	$\mathcal{K}$: {$\mathcal{K}$}
$\mathcal{B}$: {$\mathcal{B}$, $\mathcal{C}$, $\mathcal{F}$}	$\mathcal{F}$: {$\mathcal{F}$}	$\mathcal{I}$: {$\mathcal{I}$, $\mathcal{H}$}	$\mathcal{L}$: {$\mathcal{L}$}
$\mathcal{C}$: {$\mathcal{C}$, $\mathcal{A}$, $\mathcal{F}$, $\mathcal{E}$}	$\mathcal{G}$: {$\mathcal{G}$, $\mathcal{K}$, $\mathcal{D}$, $\mathcal{J}$, $\mathcal{H}$, $\mathcal{I}$}	$\mathcal{J}$: {$\mathcal{J}$, $\mathcal{G}$}	$\mathcal{M}$: {$\mathcal{M}$, $\mathcal{C}$}
$\mathcal{D}$: {$\mathcal{D}$, $\mathcal{C}$, $\mathcal{G}$}			

Die daraus errechenbare Distanzmatrix (Schritt 5) läßt sich z.B. mittels des Spring-Embedders in eine dreidimensionale Darstellung überführen, die dann anschließend wiederum mit den in Abschnitt 8.3.1 beschriebenen Parametern (insbesondere der gleichen Größe aller darzustellenden Objekte) innerhalb eines VRML-Clients darstellbar und explorierbar ist. Abbildung 47 stellt einen Screenshot dieser Visualisierung des in Abbildung 46 aufgeführten Beispielsystems dar.

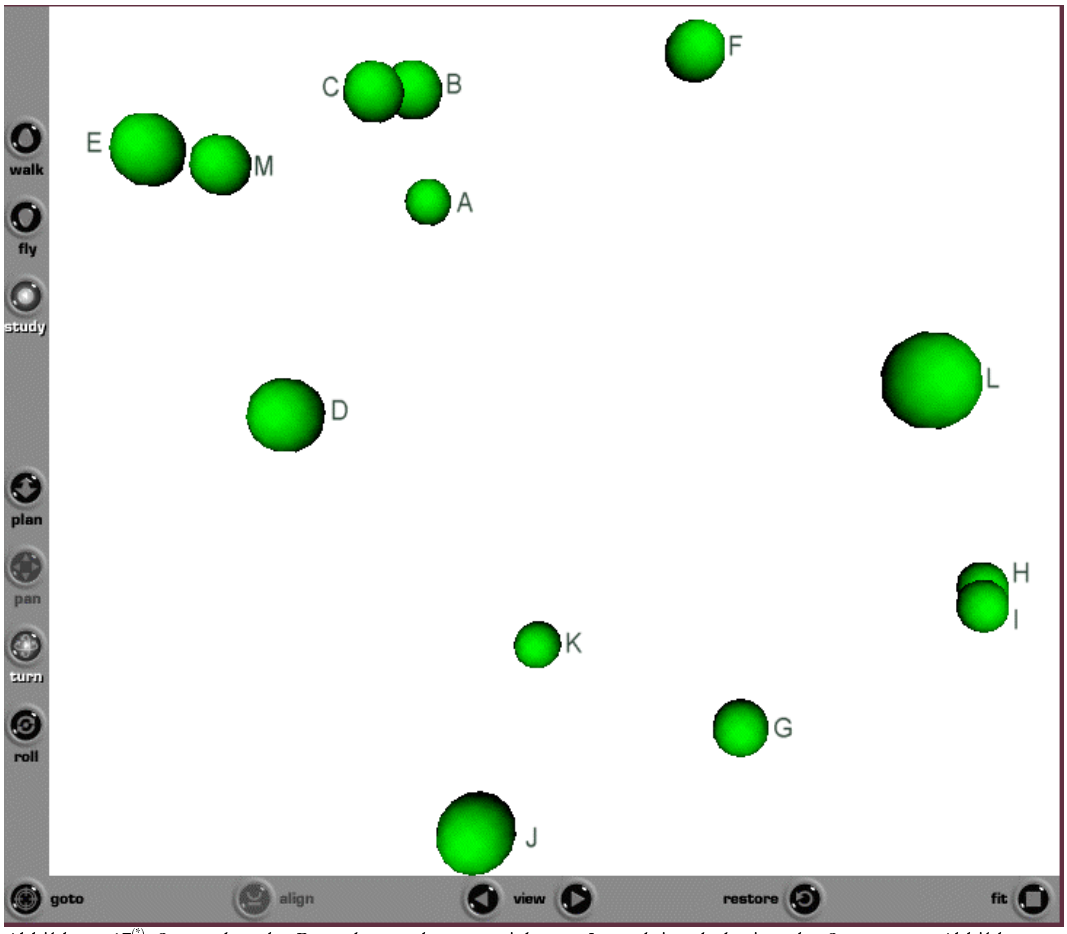


Abbildung 47(*): Screenshot der Betrachtung der ungewichteten Interaktionskohäsion des Systems aus Abbildung 46

Innerhalb dieser Visualisierung entsprechen nur 74% der dargestellten Ordnungsrelationen denjenigen innerhalb der zugrundeliegenden Distanzmatrix. Trotzdem lassen sich deutlich die folgenden Aussagen aus der Visualisierung, die vom Quelltextparsen bis zur Erstellung der dreidimensionalen Welt wiederum vollständig automatisiert abläuft, ableiten und anhand des zugrundeliegenden Systems aus Abbildung 46 validieren:

- Es existieren zwei deutlich voneinander zu unterscheidende Gruppen von Klassen, deren Trennlinie in der oben dargestellten Abbildung von links unten nach rechts oben verläuft: So existiert eine Klassengruppe G_1 mit den Klassen $\{\mathcal{H}, \mathcal{I}, \mathcal{G}, \mathcal{K}, \mathcal{J}\}$ und eine Klassengruppe G_2 mit den Klassen $\{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{F}, \mathcal{M}, \mathcal{E}, \mathcal{D}\}$. Diese zwei Gruppen interagieren über die Klasse $\mathcal{D}$, die zwischen den beiden Gruppen positioniert ist.
- Die exponierte Stellung von $\mathcal{L}$ (sehr weit im Vordergrund) demonstriert, daß $\mathcal{L}$ weder ausgehende Interaktionskopplungen besitzt, noch von anderen Klassen verwendet wird.
- Die Klassen $\mathcal{I}$ und $\mathcal{H}$ besitzen eine maximale Ähnlichkeit zueinander bzgl. der Menge von Klassen, von denen sie etwas verwenden: $\mathcal{I}$ besitzt ausschließlich ausgehende Interaktionskopplungen zu $\mathcal{H}$, das selbst keinerlei ausgehende Interaktionskopplungen besitzt.
- Die Klassen $\mathcal{B}$ und $\mathcal{C}$ besitzen ebenfalls bzgl. der Menge von Klassen, von denen sie etwas verwenden, eine hohe Ähnlichkeit zueinander: Beide verwenden $\mathcal{F}$ und es existiert eine Interaktionskopplung zwischen $\mathcal{B}$ und $\mathcal{C}$.
- Entgegen der Positionierung innerhalb des klassischen Klassendiagramms (vgl. Abbildung 46) sind auch die Klassen $\mathcal{M}$ und $\mathcal{E}$ bzgl. ausgehender Interaktionskopplungen ähnlich, da beide ausschließlich $\mathcal{C}$ verwenden.

Der Vorteil dieser dreidimensionalen, auf Klassenebene abstrahierten Aufrufgraphen liegt in der Möglichkeit, auch für sehr große Systeme anwendbar zu sein (vgl. Relevanz-Problem). Klassische Klassendiagramme sind dafür i.A. völlig ungeeignet.

Gewichtete Interaktion
Innerhalb der Erweiterung der Interaktionskohäsion um eine Gewichtung ist es sinnvoll, die zwei Kopplungsarten datenbasierte/strukturelle Kopplung und externe Kopplung separat zu betrachten (vgl. Abschnitt 8.3.2). Eine mögliche kombinierte Betrachtung, innerhalb derer z.B. direkte Attributsbenutzungen eine andere Gewichtung als direkte Methodenbenutzungen erhalten, ist darauf aufbauend dann einfach möglich.

Gewichtete Interaktion: Externe Kopplung
Diese Betrachtungsweise reduziert Kopplung zwischen zwei Klassen auf die direkte Verwendung von öffentlichen Attributen untereinander. Für den Schritt 4 – die Identifikation der die jeweilige Kopplungsart begründenden Merkmale – folgt daraus, daß die Merkmale einer Klasse C durch die Menge derjenigen öffentlichen Attribute gegeben sind, die C anbietet oder irgendeine Methode aus C von anderen Klassen verwendet. Damit ergibt sich die in Definition 39 angegebene *gewichtete, attributbasierte Interaktionskohäsion*:

> *Konstruktion der gewichteten, attributbasierten Interaktionskohäsion*: Mehrere Klassen gehören entsprechend der gewichteten, attributbasierten Interaktionskohäsion um so mehr zusammen, um so mehr gemeinsame öffentliche Attribute (inkl. der von ihnen selbst angebotenen) sie jeweils verwenden.
>
> *Merkmalsmenge der gewichteten, attributbasierten Interaktionskohäsion*: Für eine Klasse C ist die Menge der zu betrachtenden Merkmale gegeben durch:
> $\mathcal{M}$ = {öffentliche Attribute von C} $\cup$
> {innerhalb von C verwendete, öffentliche Attribute anderer Klassen}.
> Die Sicht auf C muß dabei deren Flatten-Version berücksichtigen, da C sowohl öffentliche Attribute als auch Benutzungen von öffentlichen Attributen erben kann.
>
> *Interpretation der gewichteten, attributbasierten Interaktionskohäsion*: Klassen mit einer geringen Distanz zueinander besitzen eine ähnliche Menge von öffentlichen Attributen, die sie verwenden und/oder anbieten. Die Ähnlichkeit kann eine gegenseitige Attributsverwendung einschließen. Aufgrund der ähnlichen, datenbasierten Kontexte, mit denen dicht beieinander liegende Klassen arbeiten, realisieren die betrachteten Klassen Funktionalität, die zum Teil auf ähnlichen Daten beruhen. Dies gilt sowohl für den Fall direkter, attributbasierter Interaktionskopplungen (z.B. Klasse $\mathcal{A}$ und Klasse $\mathcal{B}$ kommunizieren über Attribute von $\mathcal{A}$ und/oder $\mathcal{B}$) als auch zwischen Klassen, die über dritte Klassen über gemeinsame Attribute miteinander kommunizieren (z.B. Klasse $\mathcal{A}$ und Klasse $\mathcal{B}$ kommunizieren über Attribute einer Klasse C).

Definition 39: Attributbasierte, gewichtete Interaktionskohäsion

Wird innerhalb des Klassendiagramms aus Abbildung 46 davon ausgegangen, daß im Falle öffentlicher Attribute diese auch von allen mit der Klasse assoziierten Klassen verwendet werden, ergeben sich folgende Merkmalsmengen:

$\mathcal{A}$: {} $\mathcal{B}$: {$a16$,$a17$,$a18$} $\mathcal{C}$: {$a16$,$a17$,$a18$} $\mathcal{D}$: {}	$\mathcal{E}$: {} $\mathcal{F}$: {$a16$, $a17$, $a18$} $\mathcal{G}$: {$a22$, $a23$, $a24$, $a28$, $a29$, $a30$}	$\mathcal{H}$: {$a22$, $a23$, $a24$} $\mathcal{I}$: {$a22$, $a23$, $a24$} $\mathcal{J}$: {$a28$, $a29$, $a30$}	$\mathcal{K}$: {} $\mathcal{L}$: {} $\mathcal{M}$: {}

Bereits an diesem Beispiel fällt die in Kapitel 6.1 angedeutete Möglichkeit auf, daß die Distanz entsprechend des generischen Distanzmaßes nur dann definiert ist, wenn die Merkmalsträgermenge nicht für zwei Entitäten inkompatibel ist. Dies ist allerdings im Beispiel für alle Klassenpaare, bei denen keine Klasse wenigstens ein öffentliches Attribut besitzt oder verwendet (z.B. das Klassenpaar ($\mathcal{A}$, $\mathcal{D}$)) der Fall. Von den dort bereits aufgeführten Möglichkeiten (vgl. Kapitel 6.3.1) wird hier die Variante der Setzung auf maximale Distanz gewählt, da zwei Klassen, die weder öffentliche Attribute besitzen, noch andere direkt verwenden, bzgl. der attributbasierten, gewichteten Interaktionskohäsion völlig unähnlich sind, d.h. maximalen Abstand zueinander besitzen.
Ein Screenshot der gewichteten, attributbasierten Interaktionskopplung des Systems aus Abbildung 46 ist in Abbildung 48 dargestellt:

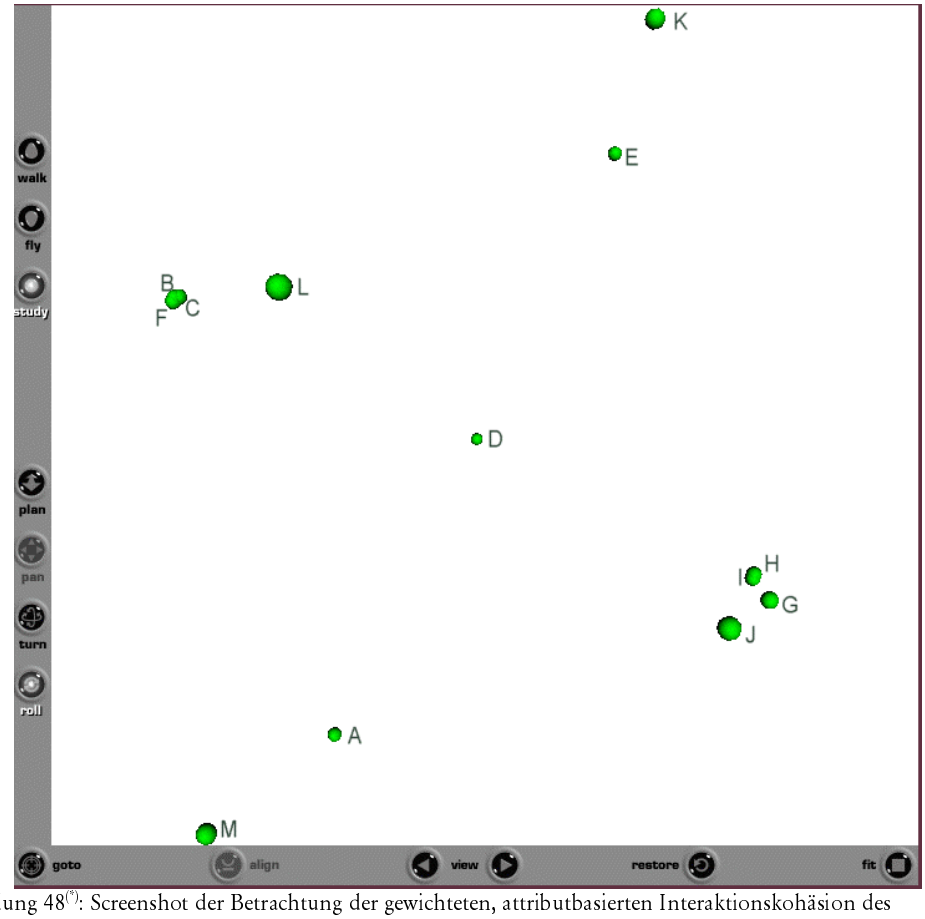


Abbildung 48(*): Screenshot der Betrachtung der gewichteten, attributbasierten Interaktionskohäsion des Systems aus Abbildung 46

Folgende Aussagen lassen sich direkt aus der Visualisierung, deren Ordnungsrelationen im Vergleich zur Distanzmatrix zu 79% erhalten sind, ableiten:

- Aufgrund der verhältnismäßig geringen direkten Attributsbenutzungen ist das Gesamtsystem sehr weit gestreut, d.h. ein Großteil der Klassen hat zueinander maximale Distanz. Derartig gestreute Visualisierungen sind typisch für Kohäsionsbetrachtungen, bei denen die kopplunggebenden Merkmale nur sehr selten vorkommen.

- Es gibt zwei Gruppen von Klassen, deren Elemente zueinander eine minimale Distanz besitzen[27]: Dies sind einerseits die Klassen $\mathcal{F}$, $\mathcal{B}$ und $\mathcal{C}$, die alle über die gleichen öffentlichen Attribute von $\mathcal{F}$ miteinander kommunizieren können, und die Klassen $\mathcal{I}$, $\mathcal{H}$, $\mathcal{G}$ und $\mathcal{J}$, bei denen $\mathcal{I}$ und $\mathcal{H}$ ebenfalls eine vollständig identische, nichtleere Menge von öffentlichen Attributen verwenden. Die Klasse $\mathcal{G}$, die zwischen den Gruppen $\mathcal{I}$, $\mathcal{H}$ und der Klasse $\mathcal{J}$ liegt, besitzt attributbasierte Interaktionskopplungen zu allen drei Klassen.

Da diese Form der Kopplung der innerhalb objektorientierter Softwareentwicklung geforderten Datenkapselung widerspricht, sind aus derartigen Ansammlungen von Klassen typische und innerhalb dieser Visualisierungen sehr einfach zu identifizierende Restrukturierungsvorschläge ableitbar (vgl. Kapitel 9).

Gewichtete Interaktion: Datenbasierte/strukturelle Kopplung
Diese Betrachtungsweise reduziert Kopplung zwischen zwei Klassen auf die direkte Verwendung von öffentlichen Methoden untereinander. Für den Schritt 4 – die Identifikation der die jeweilige Kopplungsart begründenden Merkmale – folgt daraus, daß die Merkmale einer Klasse $\mathcal{C}$ durch die Menge derjenigen öffentlichen Methoden gegeben sind, die von $\mathcal{C}$ angeboten oder verwendet werden. Damit ergibt sich die in Definition 40 angegebene *gewichtete, methodenbasierte Interaktionskohäsion*:

> *Konstruktion der gewichteten, methodenbasierten Interaktionskohäsion*: Mehrere Klassen gehören entsprechend der gewichteten, methodenbasierten Interaktionskohäsion um so mehr zusammen, um so mehr gemeinsame öffentliche Methoden (inkl. der von ihnen selbst angebotenen) sie jeweils verwenden.
>
> *Merkmalsmenge der gewichteten, methodenbasierten Interaktionskohäsion*: Für eine Klasse $\mathcal{C}$ ist die Menge der zu betrachtenden Merkmale gegeben durch:
> $\mathcal{M}$ = {öffentliche Methoden von $\mathcal{C}$} $\cup$
> {innerhalb von $\mathcal{C}$ verwendete, öffentliche Methoden anderer Klassen}.
> Die Sicht auf $\mathcal{C}$ muß dabei deren Flatten-Version berücksichtigen, da $\mathcal{C}$ sowohl öffentliche Methoden als auch Benutzungen von öffentlichen Methoden erben kann.
>
> *Interpretation der gewichteten, methodenbasierten Interaktionskohäsion*: Klassen mit einer geringen Distanz zueinander besitzen eine ähnliche Menge von öffentlichen Methoden, die sie verwenden und/oder anbieten. Die Ähnlichkeit kann eine gegenseitige Methodenverwendung einschließen. Aufgrund der ähnlichen, funktionalen Kontexte, mit denen dicht beieinander liegende Klassen arbeiten, realisieren die Klassen eine ähnliche Funktionalität (im Fall keiner direkten Interaktionskopplungen zwischen den Klassen) oder implementieren in Form von auf Klassen aufgeteilten Teilfunktionalitäten eine größere, auf höherem Abstraktionsniveau liegende Funktionalität (im Fall vieler direkter Interaktionskopplungen zwischen den Klassen).

Definition 40: Gewichtete, methodenbasierte Interaktionskohäsion

Wird innerhalb des Klassendiagramms aus Abbildung 46 davon ausgegangen, daß im Falle einer Assoziation von Klasse $\mathcal{A}$ zu Klasse $\mathcal{B}$ alle eventuell in $\mathcal{B}$ vorkommenden, öffentlichen Methoden verwendet werden, ergeben sich für die einzelnen Klassen folgende Merkmale:

[27] Führt das Ergebnis einer Dimensionsreduktion auf identische Koordinaten für unterschiedliche Entitäten, so wie dies für Klassen mit einer Distanz von 0 zueinander der Fall ist, so werden die Punkte künstlich minimal verschoben, um als unterschiedliche, aber sehr eng beieinander liegende Entitäten abgebildet zu werden.

$\mathcal{A}$: {*m*1, *m*2, *m*3, *m*4, *m*5,*m*6, *m*37, *m*38,*m*39} $\mathcal{B}$: {*m*4,*m*5,*m*6, *m*7,*m*8,*m*9} $\mathcal{C}$: {*m*7,*m*8,*m*9,*m*13, *m*14,*m*15,*m*1, *m*2,*m*3} $\mathcal{D}$: {*m*10,*m*11,*m*12, *m*7,*m*8,*m*9, *m*19,*m*20,*m*21}	$\mathcal{E}$: {*m*13,*m*14,*m*15, *m*7,*m*8,*m*9} $\mathcal{F}$: {} $\mathcal{G}$: {*m*19,*m*20,*m*21, *m*28,*m*29,*m*30, *m*25,*m*26,*m*27, *m*31,*m32*,*m*33, *m*10,*m*11,*m*12}	$\mathcal{H}$: {} $\mathcal{I}$: {*m*25,*m*26,*m*27} $\mathcal{J}$: {*m*28,*m*29,*m*30, *m*19,*m*20,*m*21}	$\mathcal{K}$: {*m*31,*m32*,*m*33} $\mathcal{L}$: {*m*34} $\mathcal{M}$: {*m*37,*m*38,*m*39, *m*7,*m*8,*m*9}

Wie bei der attributbasierten Interaktionskohäsion kann die Situation der merkmalinkompatiblen Distanzbestimmung entstehen (im obigen Beispiel z.B. für die Distanz zwischen der Klasse $\mathcal{F}$ und $\mathcal{H}$). Auch hier werden derartige Distanzen auf die maximale Distanz gesetzt, da zwei Klassen, die weder öffentliche Methoden besitzen, noch andere direkt verwenden, bzgl. der gewichteten, methodenbasierten Interaktionskohäsion völlig unähnlich sind, d.h. maximalen Abstand zueinander besitzen.
Ein Screenshot der gewichteten, methodenbasierten Interaktionskohäsion des Systems aus Abbildung 46 ist unter Verwendung derselben Parameter in Abbildung 49 dargestellt; für eine einfachere Validierung wurden zusätzlich die konkreten Assoziationen eingefügt[28].

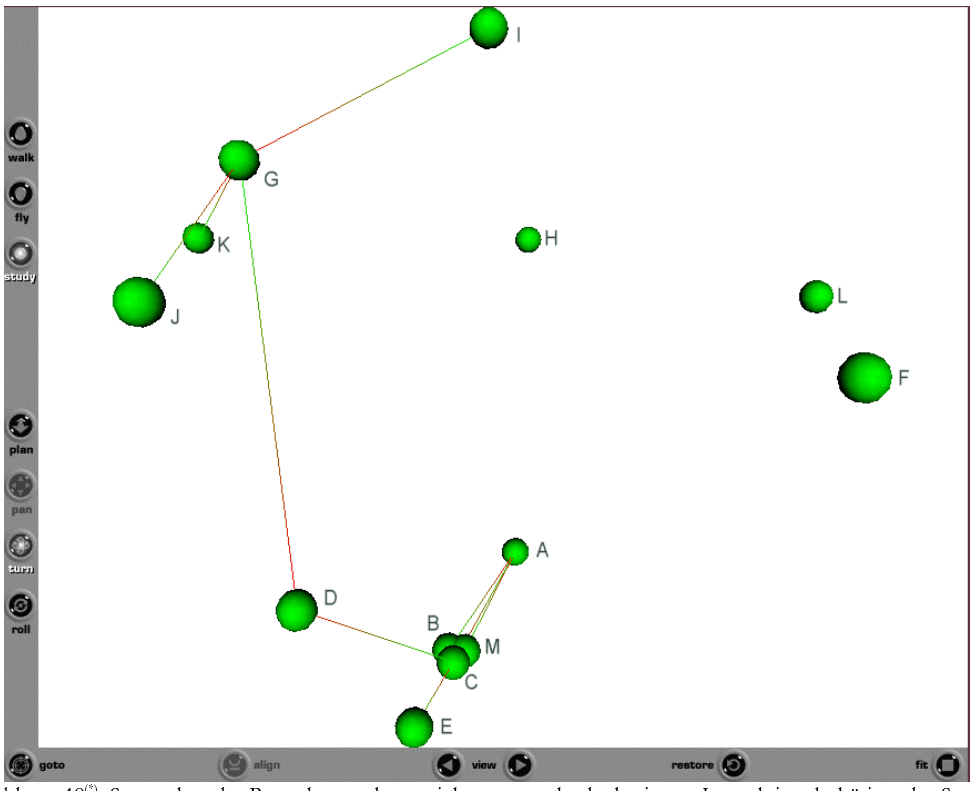


Abbildung 49(*): Screenshot der Betrachtung der gewichteten, methodenbasierten Interaktionskohäsion des Systems aus Abbildung 46

[28] Die Visualisierung vermittelt die Richtung der Kopplung durch entsprechende Farbverläufe (vgl. Kapitel 9.1), für deren Betrachtung besonders die entsprechende Abbildung im Anhang geeignet ist.

Auch hier lassen sich wieder viele entlang der Ausgangsdaten aus Abbildung 46 validierbare Aussagen direkt aus der Visualisierung, deren Ordnungsrelationen im Verhältnis zur Distanzmatrix zu 77% erhalten sind, ableiten:

- Das durch die drei Eckpunkte $\mathcal{J}$, $\mathcal{I}$, $\mathcal{K}$ aufgespannte gleichwinklige Dreieck, innerhalb dessen $\mathcal{G}$ den Mittelpunkt darstellt, belegt die zentrale Rolle der Klasse $\mathcal{G}$ und demonstriert außerdem ihre Vermittlerrolle über $\mathcal{D}$ mit der anderen Klassenmenge.
- Klassen ohne eigene öffentliche Methoden, die auch keinerlei fremde Methoden aufrufen, werden innerhalb dieser Kohäsionsbetrachtung als völlig unkohäsiv zu allen anderen Elementen dargestellt. Für dieses Beispiel gilt das für die Klassen $\mathcal{H}$ (sehr tief im Bild) und $\mathcal{F}$ (sehr weit vorne rechts im Bild). Innerhalb dieser Betrachtung fallen ebenfalls sogenannte, bzgl. der Methodeninteraktion *tote Klassen* auf, die öffentliche Methoden anbieten, die von niemandem verwendet werden (vgl. Klasse $\mathcal{L}$).
- Deutlich fallen die dicht beieinander liegenden Klassen $\mathcal{B}$, $\mathcal{C}$ und $\mathcal{M}$ auf, die bzgl. der Methodenaufrufe sehr kohäsiv zueinander sind: sie alle kommunizieren mit allen Methoden der Klasse $\mathcal{C}$. Die Klasse $\mathcal{A}$ dagegen, die weit hinten dargestellt ist, kommuniziert lediglich mit den Klassen $\mathcal{M}$ und $\mathcal{B}$, die beide nicht von der Klasse $\mathcal{C}$ verwendet werden, und wird daher näher zu diesen beiden Klassen dargestellt (die Klassen $\mathcal{M}$, $\mathcal{B}$ und $\mathcal{A}$ liegen alle hinter der Klasse $\mathcal{C}$). Die Klasse $\mathcal{E}$ dagegen ist kohäsiver zur Klasse $\mathcal{C}$ als zu den Klassen $\mathcal{B}$ und $\mathcal{M}$, da $\mathcal{E}$ ausschließlich externe Methodeninteraktionen zu $\mathcal{C}$ besitzt und $\mathcal{B}$ und $\mathcal{M}$ jeweils viel Funktionalität anbieten, die $\mathcal{E}$ nicht verwendet.

In Kapitel 9 werden für die verschiedenen Arten der klassenkopplungbasierten Kohäsion einige Prozesse, innerhalb derer die verschiedenen Kohäsionsbetrachtungen zum Einsatz kommen können, vorgestellt und anhand praktischer Erfahrungen mit großen Industrieprojekten evaluiert.

8.4 Attributs- und methodenkopplungbasierte Kohäsion

Als ein weiteres Beispiel einer Anwendung des Prozesses aus Kapitel 7.3.1 und als Vervollständigung der allgemeinen Kohäsionsbetrachtung auf unterschiedlichen Abstraktionsniveaus eines Systems (vgl. Abschnitt 8.1) werden in diesem Abschnitt die Kopplungen zwischen Methoden und Attributen betrachtet, d.h. das Ergebnis des Schritts 2 – die Bestimmung derjenigen Hierarchiestufe, deren Elemente für die Kohäsion der darüber liegenden Stufen relevant sind – besteht aus Methoden und Attributen.
Bzgl. des geforderten nächsten Schritts – die Festlegung der Kopplungsart, die zwischen den Elementen betrachtet werden soll – ist auf dieser Ebene lediglich die Methodenbenutzung und Attributsbenutzung sinnvoll einsetzbar; Vererbung bestimmt für das Verhalten innerhalb einer Klasse lediglich, welche Attribute und Methoden betrachtet werden müssen. Da die Kohäsion einer Klasse sich auf alle von ihr zur Verfügung gestellten Methoden und Attribute beziehen sollte (da sich anderenfalls evtl. die die Zusammengehörigkeit innerhalb einer Klasse bildenden Elemente auf verschiedene Oberklassen verteilen), werden für die attributs- und methodenkopplungbasierte Kohäsion die Klassen inkl. der an sie geerbten Funktionalität betrachtet (vgl. Abschnitt 8.2.1).
Da für die Kohäsion einer Klasse die Methoden- und Attributsbenutzungen sehr eng miteinander verknüpft sind (vgl. z.B. die unterschiedlichen LCOM-Varianten in Kapitel 7.1.2), werden sie im folgenden gleichzeitig betrachtet. Daraus ergibt sich die Notwendigkeit, die Merkmalsmengen für Methoden und Attribute gleichzeitig zu definieren: Während dies für Methoden in Anlehnung an die gewichtete attributbasierte und methodenbasierte Interaktionskohäsion (vgl. Abschnitt 8.3.2) geschehen kann, ist für Attribute die nach außen gerichtete Kopplung stets leer, da ein Attribut als Datum keine Implementierung besitzen kann, die Zugriff auf andere Methoden oder Attribute realisiert. Im Fall der Zuweisung eines Attributes a_1 an ein anderes Attribut a_2 innerhalb

einer Methode m_1 wird die Benutzung von a_1 und a_2 der Methode m_1 zugeordnet, da dort die entsprechende Anweisungsfolge existiert, d.h. obwohl das Attribut a_1 das Attribut a_2 verwendet, wird dies nicht entsprechend dargestellt. Um dennoch eine Zusammengehörigkeit zwischen Attributen begründen zu können, werden bei Attributen nicht die ausgehenden Kopplungen als Merkmale, sondern die eingehenden Kopplungen in Form der das Attribut benutzenden Methoden verwendet. Im Beispiel folgt daraus eine Zusammengehörigkeit der beiden Attribute a_1 und a_2, da beide innerhalb derselben Methode verwendet werden.

Damit ergibt sich die in Definition 41 angegebene methoden- und attributbasierte Interaktionskohäsion [SiStLe01]:

Konstruktion der methoden- und attributbasierten Interaktionskohäsion: Mehrere Methoden und Attribute gehören entsprechend der methoden- und attributbasierten Interaktionskohäsion um so mehr zusammen, um so mehr gemeinsame Methoden und Attribute sie verwenden (im Falle von Methoden) bzw. von um so mehr gemeinsamen Methoden sie verwendet werden (im Falle von Attributen).

Merkmalsmenge der methoden- und attributbasierten Interaktionskohäsion:
Für eine Methode m ist die Menge der zu betrachtenden Merkmale gegeben durch:
$\mathcal{M}_m = \{m\} \cup \{\text{direkt benutzte Methoden von } m\} \cup \{\text{benutzte Attribute von } m\}$.
Für ein Attribut a ist die Menge der zu betrachtenden Merkmale gegeben durch:
$\mathcal{M}_a = \{a\} \cup \{\text{Methoden, die } a \text{ benutzen}\}$.
Für die Klassenkohäsion müssen dabei innerhalb einer Klasse alle Methoden und Attribute betrachtet werden, d.h. geerbte Eigenschaften müssen gleichwertig zu neu implementierten berücksichtigt werden.

Interpretation der methoden- und attributbasierten Interaktionskohäsion: Methoden mit einer geringen Distanz zueinander besitzen eine ähnliche Menge von Methoden oder Attributen, die sie verwenden. Dies schließt eine gegenseitige Benutzung zwischen Methoden mit ein. Attribute mit einer geringen Distanz zueinander besitzen eine ähnliche Menge von Methoden, die sie verwenden. Methoden und Attribute mit einer geringen Distanz zueinander deuten auf eine Benutzung des Attributs durch die Methode und eine verhältnismäßig geringe Benutzung des Attributs durch andere Methoden hin. Für einen Cluster von jeweils eng beieinander liegenden Methoden und Attributen bedeutet dies: Aufgrund der ähnlichen funktionalen Kontexte, mit denen dicht beieinander liegende Methoden arbeiten, realisieren diese Elemente zusammen eine gemeinsame Funktionalität, die auf den ebenfalls gemeinsam verwendeten Attributen basiert. Solche Cluster stellen die Idealvorstellung eines abstrakten Datentyps als Daten mit dazugehörigen Operationen dar.

Definition 41: Methoden- und attributbasierte Interaktionskohäsion

Da für diese sehr feingranulare Betrachtung einer Klasse innerhalb der UML keine Standardnotation vorgesehen ist, wird für diese Kohäsionsbetrachtung ein Beispielquelltext angegeben, der aus 2 Klassen mit jeweils 3 Methoden und 2 Attributen besteht (vgl. [SiStLe01]); dabei sind alle Methoden und Attribute als `static` deklariert, um auf Objekt-Instanziierungen der einzelnen Klassen in diesem Beispiel verzichten zu können:

```
class class_A
{
public:
      static void methodA1()
      {     attributeA1=0;
            methodA2();
      }
      static void methodA2()
      {     attributeA2=0;
            attributeA1=0;
      }
      static void methodA3()
      {     attributeA1=0;
            attributeA2=0;
            methodA1();
            methodA2();
      }
      static int attributeA1;
      static int attributeA2;
}
```

```
class class_B
{
public:
      static void methodB1()
      {     class_A::attributeA1=0;
            class_A::attributeA2=0;
            class_A::methodA1();
      }
      static void methodB2()
      {     attributeB1=0;
            attributeB2=0;
      }
      static void methodB3()
      {     attributeB1=0;
            methodB1();
            methodB2();
      }
      static int attributeB1;
      static int attributeB2;
}
```

Nach der Erstellung der Distanzmatrix (Schritt 5) kann mit der Visualisierung begonnen werden. Da innerhalb dieser Kohäsionsbetrachtung zwei unterschiedliche Arten von Entitäten, nämlich Attribute und Methoden, gleichzeitig dargestellt werden müssen, wird für die folgende Visualisierung der Freiheitsgrad der Form wie folgt festgelegt: Methoden werden als Kugeln und Attribute als Würfel dargestellt. Alle Elemente einer Klasse besitzen dabei eine identische Farbe. Mit diesen Parametern ist für das obige Beispiel die in Abbildung 50 dargestellte Visualisierung möglich. Folgende Aussagen lassen sich direkt aus dieser Visualisierung ableiten (und entlang des oben dargestellten Quelltextes validieren):

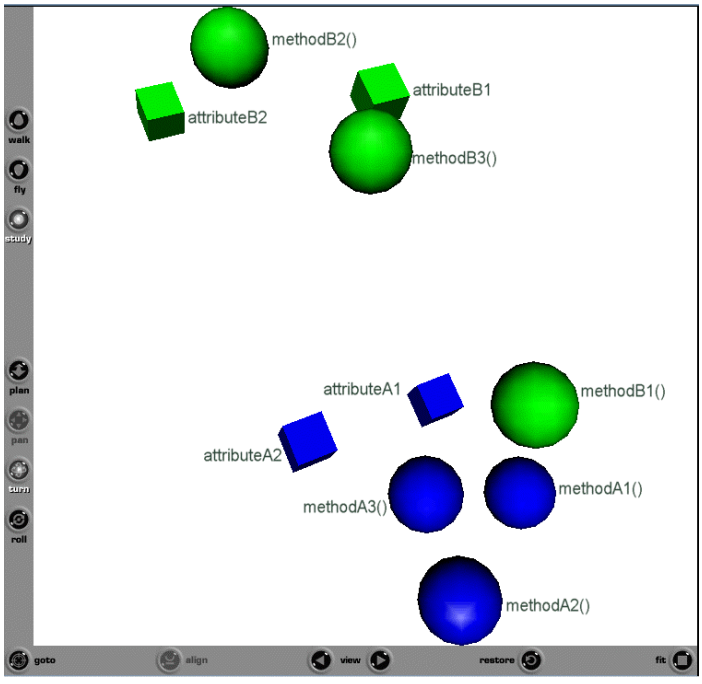


Abbildung 50[(*)]: Screenshot der Betrachtung der methoden- und attribut-basierten Interaktionskohäsion des Beispielsystems (s.o.)

- Sowohl die Attribute und Methoden `attributeB1`, `attributeB2`, `methodB2()`, `methodB3()` (oberer Cluster) als auch die Methoden und Attribute `attributeA1`, `attributeA2`, `methodA1()`, `methodA2()`, `methodA3()`, `methodB1()` (unterer Cluster) bilden jeweils einen kohäsiven Cluster, der als ADT-Realisierung angesehen werden kann.
- Die exponierte Stellung der Methode `methodB1()` innerhalb des unteren Clusters belegt, daß diese Methode mehr aus der Klasse `class_A` als aus der Klasse `class_B` benutzt.
- Die Methode `methodB3()` stellt über die Methode `methodB1()` eine Verbindung zum oberen Cluster dar.

Auch diese Art der Kohäsionsbetrachtung ist im Kapitel 9 in einen Prozeß eingebettet, der anschließend entlang einiger Beispielprojekte für die Evaluation dieser Kohäsionsbetrachtung verwendet wird.

8.5 Allgemeine merkmalbasierte Kohäsion

Im Gegensatz zu den kopplungbasierten Kohäsionsbetrachtungen objektorientierter Systeme, die einerseits die besonderen Arten von Kopplungen und andererseits ihre Abhängigkeiten vom Konzept der Vererbung berücksichtigen, sind die allgemeinen merkmalbasierten Kohäsionsbetrachtungen weniger spezifisch: Lediglich die vorgestellten Abstraktionsstufen (vgl. Abschnitt 8.1) geben Anhaltspunkte, welche Ebenen primär für eine Merkmalsextraktion relevant sind.
Während die für eine kopplungbasierte Kohäsionsbetrachtung relevanten Daten dem Quelltext, d.h. den in einer Programmiersprache abgefaßten Anweisungen entnommen werden, die aufgrund der Anforderungen durch die jeweils verwendeten Compiler eine detaillierte und wohlstrukturierte Syntax besitzen, ist die Extraktion allgemeiner Merkmale, die nicht zwangsläufig für das Laufzeitverhalten des Quelltextes relevant sind, deutlich schwieriger. Folgende Problemklassen können dabei identifiziert werden:

- *Mehrdeutigkeit*: Das häufige Fehlen automatischer Syntaxchecks führt dazu, daß viele Merkmale mehrdeutig zu interpretieren sind. So ist z.B. die Angabe des Erstellungsdatums innerhalb einer Klasse in verschiedenen Formaten möglich, die nicht immer eindeutig zu unterscheiden sind; so kann die Angabe 01/11/2000 entsprechend der deutschen Schreibweise für den 1. November, entsprechend der amerikanischen Schreibweise aber für den 11. Januar gehalten werden.
- *Inkonsistenz*: Eng verbunden mit dem Problem der Mehrdeutigkeit ist das Problem der Inkonsistenz: So kann ein und derselbe Autor sich in unterschiedlichen Klassen durch unterschiedliche Namensangaben als Autor zu erkennen geben: dies äußert sich häufig durch unterschiedliche Schreibweisen (z.B. durch den unterschiedlichen Umgang mit Umlauten: ‚ä' oder ‚ae'), unterschiedliche Stile (z.B. durch das Wechseln der Reihenfolge von <Vorname> und <Nachname>), variierende Abkürzungen (z.B. durch das Verwenden von Initialen) und semantisch begründete Synonyme (z.B. <Name>=Projektleiter=Hauptverantwortlicher etc.).
- *Unvollständigkeit:* Da diese Merkmale nicht direkt für das Funktionieren des Softwaresystems relevant sind, werden sie häufig weggelassen (z.B. keine Angabe über den Autor) oder unbewußt gesetzt (z.B. durch von Werkzeugen gesetzte Standardwerte).

Diese Probleme führen dazu, daß die merkmalbasierte Kohäsionsbetrachtung häufig nicht vollständig automatisiert durchgeführt werden kann, da Werkzeuge i.d.R. nur nach syntaktischen Mustern suchen können, diese aber i.d.R. noch nachbearbeitet werden müssen. Hilfreich sind in diesem Zusammenhang fest vorgegebene Kodierrichtlinien (*Coding Conventions*), da sie durch ihre Regeln ein einheitlicheres und damit einfacher automatisch zu extrahierendes Erscheinungsbild der Merkmale fordern. Ein Beispiel solcher Kodierrichtlinien sind die *JAVA-Code-Conventions* (vgl. z.B. [CWHT98]), in denen eine *JAVADOC*-kompatible Kommentierung vorgesehen ist. JAVADOC ist ein Werkzeug, das für relevante Kommentare eine spezielle Syntax mit expliziten Markierungen, sogenannten *Tags,* vorsieht, die es dem Werkzeug später ermöglicht, eine mit eben diesen durch die Tags erkennbaren Kommentierungen angereicherte html-Struktur von JAVA-Quelltexten zu erstellen. So ist es neben Standardeinträgen wie z.B. der Autorenangabe (Tag: `@author`), der Versionsangabe (Tag: `@version`, wobei die Versionsangabe selbst in SCCS-Notation erfolgen soll) oder der Angabe von Verweisen auf zusätzliche Dokumentationen (Tag: `@see`) ebenfalls möglich, eigene Tags dem Werkzeug bekannt zu machen, so daß durch sie

markierte Angaben ebenfalls extrahiert werden; dies wird zur Zeit z.B. häufig verwendet für die Tags Kategorie (Tag: `@cat`) zur Klassifizierung einzelner Softwareteile, BeanInfo (Tag: `@Beaninfo`) zur Angabe weiterer Informationen einer Klasse hinsichtlich ihrer Verwendung von JavaBeans und Beispiel (Tag: `@example`) zur Angabe einer Beispieldatei, die für das Verständnis der gerade betrachteten Softwareteile relevant ist.
Im folgenden soll als ein typisches Beispiel einer merkmalbasierten Kohäsionsbetrachtung die Autorenschaft einer Klasse entsprechend des Prozesses aus Kapitel 7.3.2 erstellt werden.
Ein mögliches Ziel einer auf der Autorenschaft von Klassen/Dateien basierenden Kohäsionsbetrachtung ist die Überprüfung, inwieweit die i.d.R. aufgrund einer Dekomposition des Problems in Teilbereiche entstandenen Subsysteme adäquat auf einzelne Projektmitglieder bzw. Projektgruppen übertragen wurden, d.h. inwieweit einzelne Projektmitglieder verantwortlich für einzelne Subsysteme sind. Der Schritt 1 des Prozesses – die Bestimmung der Hierarchiestufe, für die Kohäsion bestimmt werden soll – ist damit auf die Subsystemebene festgelegt; der Schritt 2 – die Bestimmung einer darunter liegenden Hierarchiestufe, deren Elemente für die Kohäsion der darüber liegenden Stufe relevant ist – ist innerhalb des Beispiels auf die Dateiebene festgelegt. Aufgrund der in JAVA üblichen Regelung, pro Datei nur eine öffentliche Klasse zu definieren, kann diese Stufe allerdings im folgenden auch problemlos auf die Klassenebene verfeinert werden. Die Bestimmung der Merkmale, die die Zusammengehörigkeit der Klassen festlegen (Schritt 3), ist durch das Auslesen des JAVADOC-Tags „`@author`“ gegeben. Die Autorenschaft selbst kann, wenn keinerlei zusätzliche Informationen über die einzelnen Autoren gegeben sind, lediglich als nominalskaliertes, mehrstufiges Merkmal aufgefaßt werden (vgl. Kapitel 7.3.2). Dabei sind folgende Ausprägungsarten möglich:

- Eine Klasse wurde von genau einem Autor verfaßt. In diesem Fall besteht die Merkmalsmenge einer Klasse aus der Menge {<Autorenname>}.
- Eine Klasse wurde von keinem Autor verfaßt: Entweder wurde die Klasse automatisch generiert und der Generator fügt keine JAVADOC-Tags ein, oder der Autor hat den entsprechenden Eintrag vergessen. Da für derart merkmallose Klassen das Distanzmaß nicht definiert ist, muß eine der in Kapitel 6.3.1 vorgestellten Setzungen angewendet werden: In diesem Fall ist es sinnvoll, Klassen, die keinerlei Angaben zur Autorenschaft besitzen, als „besonders“ zu kennzeichnen: Damit weisen diese besonderen Klassen zu anderen Klassen mit einer Autorenangabe eine große Distanz auf, die verschiedenen autorenlosen Klassen besitzen untereinander aber eine sehr geringe Distanz. Eine Möglichkeit, diese Setzung zu erreichen, ist z.B., die Merkmalsmenge autorenloser Klassen mit dem Element {„anonym“} festzulegen.
- Eine Klasse wurde von mehreren unterschiedlichen Autoren verfaßt: In diesem Fall besteht die Merkmalsmenge einer Klasse aus der Menge {<Autorenname$_1$>, ..., <Autorenname$_n$>}

Nach der Extraktion dieser Merkmale und einer manuellen Nachbearbeitung (vgl. oben) kann die Distanzmatrix erstellt und anschließend visualisiert werden. In der folgenden Abbildung 51 ist eine Beispielvisualisierung (inkl. nachträglich eingefügter Annotationen) eines Systems mit 642 Klassen dargestellt.

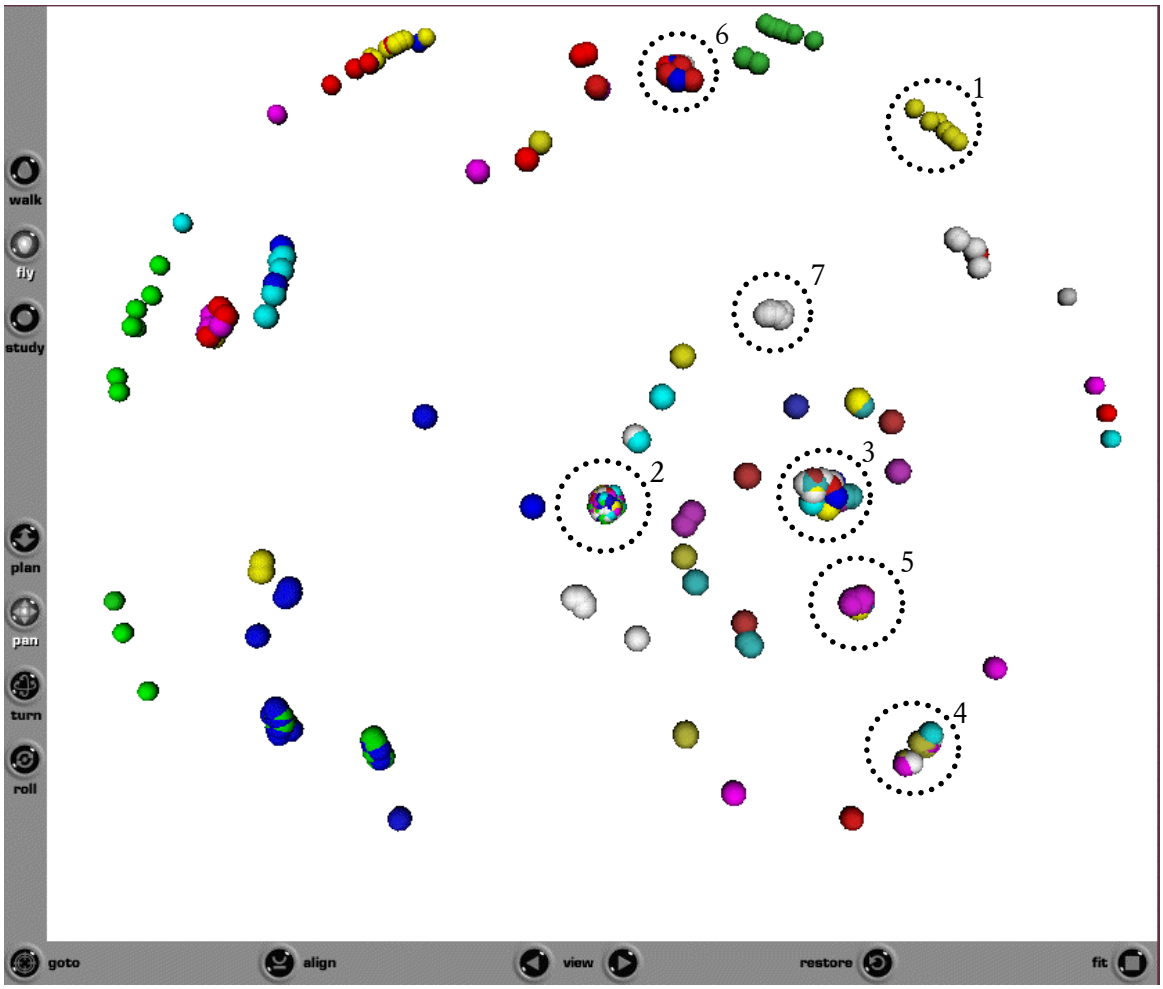


Abbildung 51(*): Screenshot der Betrachtung der autorenbasierten Kohäsionsbetrachtung

Die Visualisierungsparameter sind dabei wie folgt gesetzt: Jede dargestellte Kugel repräsentiert eine Klasse, jede Kugel hat eine Einheitsgröße und die Farbe ist entsprechend der Subsystemzugehörigkeit gewählt, d.h. Klassen aus demselben Subsystem besitzen dieselbe Farbe.

Die einzelnen Ausprägungsarten dieser Merkmale lassen sich in der Visualisierung deutlich wiederfinden:

- Werden mehrere Klassen von ein und demselben Autor geschrieben, so formieren sich diese Klassen zu einem Cluster zusammen. Ein Beispiel einer solchen Klassenmenge ist im Cluster 1 dargestellt.
- Klassen ohne Autorenangabe sind deutlich von den anderen Klassen getrennt, bilden aber untereinander wieder ein eng zusammengehöriges Cluster. In Abbildung 51 ist diese Menge durch das Cluster 2 sehr tief im Bild repräsentiert.
- Werden mehrere Klassen von derselben Menge von Autoren geschrieben, so formieren sich diese Klassen zu einem Cluster zusammen (vgl. z.B. Cluster 5). Der Cluster 5 liegt dabei in der Mitte derjenigen Cluster, innerhalb derer die Autorenschaft der Klassen ausschließlich durch eine echte, nicht-leere Untermenge der Autoren vom Cluster 5 gegeben ist. So ist z.B. der Cluster 5 in Abbildung 51 durch diejenige Klassen gegeben, die von zwei Autoren $\mathcal{A}_1$ und $\mathcal{A}_2$ gleichzeitig verfaßt worden sind. Cluster 5 liegt dabei in der Mitte von Cluster 4, innerhalb dessen die Klassen vom Autor $\mathcal{A}_1$ geschrieben worden sind, und Cluster 3, dessen

Cluster vom Autor $\mathcal{A}_2$ verfaßt worden sind. Ein ähnliches Muster, das jedoch drei verschiedene Autoren abbildet, ist durch die Gruppen der Einzelautoren Cluster 3, 1 und 6 gegeben. Das Cluster 7, das zudem lediglich aus Klassen eines Subsystems besteht (gleiche Farbe), ist von allen drei Autoren zusammen erstellt wurden.

Auch diese, eher managementorientierte Sicht kann wertvolle Einblicke in große Softwaresysteme ermöglichen. Die Mächtigkeit der allgemeinen, kopplungbasierten Kohäsion hängt dabei wesentlich von dem Grad der Automatisierbarkeit der Merkmalsextraktion ab; nur wenn der Aufwand der i.d.R. notwendigen manuellen Nachbearbeitung nicht zu hoch ist, kann diese Kohäsionsbetrachtung effizient eingesetzt werden.

8.6 Zusammenfassung

Objektorientierte Systeme bieten aufgrund ihrer i.d.R. explizit gemachten Struktur und der damit einfacheren Möglichkeit, mittels Werkzeugen informationsreiche Produktmodelle zu erstellen, reichhaltige Möglichkeiten einer Kohäsionsbetrachtung: Die wesentlichen Abstraktionsniveaus, auf denen die Kohäsion betrachtet werden kann, sind durch die hierarchische Struktur von System, Subsystemen, Dateien, Klassen, Methoden und Attributen gegeben. Entsprechend der zwei Prozesse zur Kohäsionsbestimmung können auf jeder dieser Ebene kopplungbasierte und allgemeine merkmalbasierte Kohäsionsbetrachtungen angestellt werden. Für die erstere spielt die Vererbung eine besondere Rolle, da die separate und kontextlose Betrachtung speziell von Klassen innerhalb einer Vererbungshierarchie ein teilweise unvollständiges Bild abgibt. Mittels der vorgestellten Flatten-Technik werden sowohl geerbte Funktionalität, geerbte Kopplungen als auch potentielle Kopplungen zu Unterklassen entsprechend berücksichtigt. Diese drei Aspekte spielen für die vier verschiedenen Arten der Vererbung Generalisierung/Spezialisierung, Schnittstellenabstraktion, Typhierarchieimplementierung und Quelltextwiederverwendung eine unterschiedlich wichtige Rolle.

Für die kopplungbasierte Kohäsion auf Hierarchiestufen über der Klassenebene wird der Prozeß exemplarisch angewendet für die

- allgemeine Vererbungskohäsion, innerhalb derer Klassen um so mehr zusammengehören, um so ähnlicher die Mengen ihrer Oberklassen sind,
- gewichtete Vererbungskohäsion, innerhalb derer Klassen um so mehr zusammengehören, um so mehr aufgrund der Vererbung identische Funktionalität sie nach außen anbieten,
- ungewichtete Interaktionskohäsion, innerhalb derer Klassen um so mehr zusammengehören, um so mehr gemeinsame Klassen (inkl. ihrer selbst) sie jeweils verwenden,
- gewichtete, attributbasierte Interaktionskohäsion, innerhalb derer Klassen um so mehr zusammengehören, um so mehr gemeinsame öffentliche Attribute (inkl. der von ihnen selbst angebotenen) sie jeweils verwenden und für die
- gewichtete, methodenbasierte Interaktionskohäsion, innerhalb derer Klassen um so mehr zusammengehören, um so mehr gemeinsame öffentliche Methoden (inkl. der von ihnen selbst angebotenen) sie jeweils verwenden.

Für die kopplungbasierte Kohäsion auf Hierarchiestufen über der Methoden/Attributsebene wird der Prozeß exemplarisch angewendet für die

- methoden- und attributbasierte Interaktionskohäsion, innerhalb derer Methoden und Attribute um so mehr zusammengehören, um so mehr gemeinsame Verwendungen oder Verwendungen zwischen ihnen existieren.

Innerhalb aller kopplungbasierten Kohäsionsbetrachtungen spielt das Flatten eine herausragende Rolle, da nur dadurch die Zusammengehörigkeit einer Entität zu einer anderen Entität vollständig beurteilt werden kann.

Für die allgemeine merkmalbasierte Kohäsion auf Hierarchiestufen über der Klassenebene wird der Prozeß exemplarisch angewendet für die

- autorenbasierte Kohäsion, innerhalb derer Klassen um so mehr zusammengehören, um so ähnlicher die Menge ihrer Autoren sind.

Alle Kohäsionsbetrachtungen stellen dabei lediglich eine Anwendung des Prozesses dar und können aufgrund der generischen Werkzeugunterstützung, die auf dem generischen Distanzmaß basiert, innerhalb einer einheitlichen Werkzeuglandschaft durchgeführt werden. Außerdem erlaubt die einheitliche Infrastruktur ein sehr leichtes Einführen neuer Kohäsionsbetrachtungen, die bzgl. eigener Vorgaben eine aussagekräftige Kohäsionsbetrachtung unterstützen.

„Was in der Theorie richtig ist,
muß auch in der Praxis zutreffen;
trifft es nicht zu, so liegt ein Fehler in der Theorie,
irgend etwas ist übersehn
und nicht in Anschlag gebracht worden,
folglich ist's auch in der Theorie falsch."
(Arthur Schopenhauer: *„Eristische Dialektik",* Kunstgriff 33)

9 Externe Validierung des generischen Kohäsionskonzepts

In Kapitel 4.3 wurde die Notwendigkeit der Validierung von Softwaremaßen und die Verfeinerung in eine interne und externe Validierung erläutert. Während die interne Validierung im wesentlichen theoretisch geschehen kann, und dies für das generische Kohäsionsmaß in den Kapiteln 6 allgemein und in den Kapiteln 7 und 8 konkret für jeweilige Instanziierungen des generischen Distanzmaßes intensiv geschehen ist, bedarf die externe Validierung der Anwendung des Maßes in realen Projekten, um die Überprüfung, ob die entsprechende Softwareprodukt-vermessung tatsächlich Fragestellungen in der realen Welt beantworten hilft, durchführen zu können (vgl. Kapitel 4.3). Da das generische Kohäsionskonzept als Erweiterung bisheriger Meßansätze (vgl. Kapitel 5) aufgefaßt werden kann, wird die Validierung im folgenden für Kombinationen knotenorientierter und kantenorientierter Maße durchgeführt. Deren Validierung soll hier entlang der Prozesse "Beurteilung von Produktqualität" (Kapitel 3.3.1) und "Überarbeitung eines Programms" (Kapitel 3.3.3) geschehen.
Der Einsatz der in dieser Arbeit vorgestellten Konzepte im Rahmen realer Projekte für eine anschließende Validierung erfordert eine zuverlässige und vor allen Dingen bzgl. der in der Praxis vorkommenden Systemgrößen skalierende Werkzeugumgebung. In Abschnitt 9.1. wird dafür eine auf dem bereits in Kapitel 3.5. vorgestellten Werkzeug Crocodile aufbauende Umgebung vorgestellt, die das Konzept der Multisicht realisiert (vgl. Kapitel 5.3.2). In Abschnitt 9.2. werden zwei typische Anwendungen dieser Werkzeuge innerhalb des Prozesses "Beurteilung von Produktqualität" für zwei große, kommerzielle Softwareprojekte vorgestellt. Der in Abschnitt 9.3 vorgestellte Werkzeugeinsatz innerhalb des Prozesses "Überarbeitung eines Programms" geschieht entlang eigener Softwareprodukte bzw. öffentlich zugänglicher Software. Daß im Zuge der gewünschten Validierung Projektpartner und damit reale Projekte primär nur für den Prozeß der Beurteilung von Produktqualität innerhalb einer Hochschulzusammenarbeit zu gewinnen sind, hat im wesentlichen folgende Gründe:

- *Managementmotivation*: Das Management, das i.d.R. über Ressourcen (und damit über mögliche Kooperationsaktivitäten) innerhalb eines Projekts entscheidet, ist i.A. weniger an technischen Details interessiert, wie sie Thema der Prozesse "Überarbeitung eines Programms" oder "Vorbereitung eines Reviews" sind, als an einer Management-orientierten Unterstützung, wie sie als Ergebnisse der Outsource-Beurteilung, Meilenstein-Beurteilung oder der Standard-Einhaltung (vgl. Kapitel 3.3.1) anfallen.
- *Technikstand*: Einige Prozesse erfordern von der Entwicklungsumgebung des Projektpartners einen gewissen Technikstand. So erfordert z.B. der Prozeß der Trendanalyse das Verwenden eines Versionsmanagementsystems, um längerfristige, auf zurückliegenden Versionen basierende Trends erkennen zu können. Das immer noch häufig anzutreffende, manuelle Versionsmanagement, bei dem alte Versionen nach erfolgreichem Testen durch neue Versionen überschrieben werden, verhindert eine Trendanalyse, da diese alle zu vermessenden Softwareteile in allen zu betrachtenden Versionen benötigt (vgl. Kapitel 3.3.4). Anderenfalls können nur zukünftige Versionen betrachtet werden.

Ähnlich restriktiv erfordert der Prozeß der Reviewvorbereitung eine Entwicklungsphilosophie, in der die organisatorischen und technischen Vorbedingungen eines Reviews vorhanden sind.

- *Geheimhaltung*: Speziell reine Softwarefirmen müssen die von ihnen erstellten Programme als das gesamte ihnen zur Verfügung stehende Vermögen betrachten. Trotz restriktivster Geheimhaltungsverträge (sog. *Non-Disclosure-Agreements*), die innerhalb dieser Arbeit die Anonymisierung vieler Daten erforderlich machen, wird der Quelltext für eine Softwarevermessung nur ungern zur Verfügung gestellt. Eine Lösung ist häufig das Beschränken der für eine Vermessung benötigten Daten auf das Produktmodell, d.h. nur das Produktmodell muß für eine Vermessung nach außen gereicht werden. Der Nachteil dieses Vorgehens ist einerseits die Notwendigkeit, die Werkzeugumgebung für eine Produktmodellextraktion innerhalb der Softwarefirma installieren zu müssen, und andererseits, sämtliche Analysen lediglich auf diesem Produktmodell durchführen zu können. Die Möglichkeit, bei Bedarf den Quelltext nach zusätzlichen Hinweisen zu durchsuchen (z.B. in Form von Kommentierungen) ist damit nicht vorhanden.
- *Psychologie*: Das Vermessen von von Menschen geschriebener Software wird häufig als Eingriff in ihre persönliche Freiheit aufgefaßt (vgl. Kapitel 3.6). Wird dennoch eine Vermessung durchgeführt (z.B. aufgrund einer Management-Entscheidung), nehmen viele Entwickler eine feindliche Haltung gegenüber jeglicher Art von Meßwerten und darauf basierenden Interpretationen ein. Eine externe Validierung innerhalb eines solchen Umfelds ist nicht mehr möglich. Diese Blockade nimmt mit dem Detaillierungsgrad der meßbasierten Analyse noch zu; daraus folgt, daß z.B. die meßwertbasierte Beurteilung der Produktqualität der meßbasierten Überarbeitung des Programms vorzuziehen ist.

Trotz dieser spezifischen Probleme, die zusätzlich zu den allgemeinen Problemen empirischer Arbeiten auftauchen (vgl. z.B. Kap. 12 in [EmDrMe98] und Kap. 3.6.1 dieser Arbeit) kann für eine Beurteilung der Gültigkeit von Maßen auf eine externe Validierung mittels Projekten aus der Praxis nicht verzichtet werden, da nur dort die entsprechenden, für die kommerzielle Softwareentwicklung typischen Umgebungsparameter existieren. Allerdings kann diese Vielzahl von Hürden ein wesentlicher Grund für die häufig kritisierte Dichotomie der Softwarevermessung in eine praktische und eine theoretische Richtung sein (vgl. [Glas96] und Kapitel 3.6); Institutionen wie z.B. das *Institut für experimentelles Software Engineering* (IESE) in Kaiserslautern oder Verbundprojekte zwischen Wirtschaft und Universität wie z.B. das SPICE-Projekt [EmDrMe98] versuchen, genau diese Lücke zu schließen.

9.1 CrocoCosmos

Entsprechend den Anforderungen an eine Multisicht (vgl. Kapitel 5.3.2) und unter Berücksichtigung der Erweiterung bisheriger Meßansätze um das generische Distanzkonzept ist mit *CrocoCosmos* im Rahmen dieser Arbeit eine stabile Werkzeugumgebung erstellt worden, die folgende Rohdaten visuell für die Durchführung des jeweiligen Prozesses aufbereitet:

- *Projekthierarchie*: Diese hierarchische Enthaltenseinsstruktur entstammt dem CrocoBrowse-Werkzeug (vgl. Kap. 3.3.2) und stellt jeweils alle Entitäten der Abstraktionsschichten Subsystem, Datei, Klasse und Methode/Attribut dar. In Abbildung 52 ist eine derartige Struktursicht, innerhalb derer jede Ebene interaktiv beliebig ein- und ausgeblendet werden

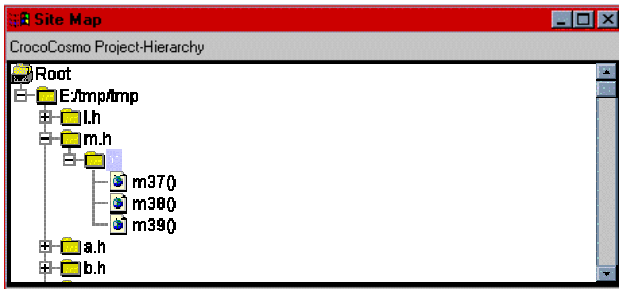


Abbildung 52: Beispiel einer Projekthierarchie innerhalb CrocoCosmos

kann, für das im Rahmen der gewichteten, methodenbasierten Interaktionskohäsion präsentierte Beispielsystem (vgl. Kapitel 8.3.2) dargestellt. Die Projekthierarchie wird vor allen Dingen während einer Exploration für die einfache Selektion interessant scheinender Softwareteile innerhalb der Kohäsionsdatenvisualisierung (s.u.) verwendet.

- *Kohäsionsdaten*: Da das generische Distanzmaß Abstände zwischen Entitäten ermittelt, die aufgrund ihrer einfachen Übertragbarkeit auf die Euklidische Distanz innerhalb geometrischer Räume hervorragend visualisierbar sind (vgl. Kapitel 6.4.2), werden diese Daten zusammen mit klassischen, knotenorientierten Daten für die Erzeugung eines dreidimensionalen, virtuellen Informationsraums verwendet (vgl. Kapitel 6.4.2), innerhalb dessen die Darstellungsparameter wie folgt verwendet werden können:
  - *Entitätsübersicht*: Alle zu betrachtenden Entitäten werden als geometrische Objekte derart dargestellt, daß die ordnungsrelativen Verhältnisse zwischen jeweils zwei Objektpaaren bestmöglich erhalten bleiben. Für diese Optimierungsaufgabe wird die Spring-Embedder-Technik verwendet (vgl. Kapitel 6.4.2). Die notwendigen Distanzen werden in dem Meßwerkzeug Crocodile errechnet. Eine einfache Schnittstelle ermöglicht das zügige Implementieren neuer, über die in Kapitel 8 vorgestellten Kohäsionsbetrachtungen hinausgehende Instanziierungen des generischen Distanzmaßes. Innerhalb der hier beschriebenen Beispielprojekte sind folgende Distanzkonzepte angewendet worden: Die allgemeine Vererbungskohäsion, die gewichtete, attributbasierte Interaktionskohäsion, die gewichtete, methodenbasierte Interaktionskohäsion und die methoden- und attributbasierte Interaktionskohäsion (vgl. Kapitel 8.3.1, 8.3.2 und 8.4).
  - *Entitätserscheinung*: Neben dieser kantenorientierten Meßwertdarstellung können die knotenorientierten Meßwerte in diesen virtuellen Informationsraum eingeblendet werden, indem die Darstellungsparameter der einzelnen dargestellten Objekte entsprechend modifiziert werden. Hier sind besonders relevant die Größe des Objekts und dessen Farbe. Die Abhängigkeit beider Parameter zu jeweils einem Maß kann während der Betrachtung dynamisch festgelegt werden. Da der menschliche Wahrnehmungsapparat Größenunterschiede sehr viel präziser wahrnehmen kann als Farbunterschiede (vgl. z.B. Kapitel 4 und 5 in [Herc94]), sollten Maße eines höheren Skalenniveaus (z.B. Rationalskala) auf die Größe und Maße eines geringeren Skalenniveaus (z.B. Nominalskala) auf die Farbe abgebildet werden. Innerhalb der Beispielprojekte ist die Farbe ausschließlich für die Angabe zur Zugehörigkeit übergeordneter Hierarchieebenen verwendet worden, d.h. durch ein nominalskaliertes, mehrstufiges Merkmal gegeben (vgl. Kapitel 7.3.2). So besitzen alle Klassen eines Subsystems dieselbe Farbe, wohingegen für Klassen unterschiedlicher Subsysteme unterschiedliche Farben verwendet werden.
- *Knotenorientierte Meßdaten*: Neben den kantenorientierten Meßdaten werden innerhalb der Beispielprojekte auch beliebige klassische, knotenorientierte Meßdaten dargestellt (zusätzlich zu den zwei knotenorientierten Maßen für die Entitätserscheinung, s.o.). Für diesen Zweck existieren zwei unterschiedliche Kontexte der Meßwerte:
  - *Entitätorientiert*: Für jede innerhalb des virtuellen Informationsraums dargestellte Entität kann eine Übersicht bzgl. der für sie gemessenen Meßwerte angezeigt werden. Dies geschieht mittels Kiviat-Diagrammen, deren Parameter wie folgt gesetzt sind (vgl. Abbildung 53): Die Auswahl der Maße

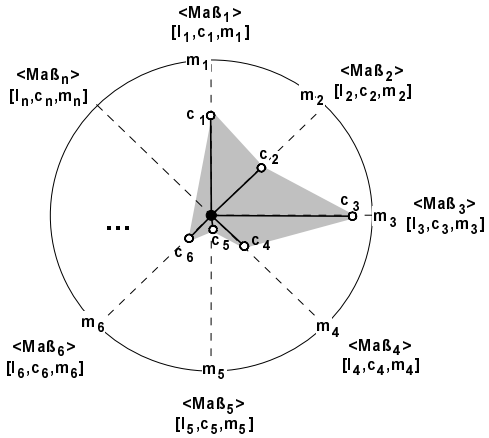


Abbildung 53: Schema eines Kiviatdiagrammes einer Entität

(<Maß$_1$>, ..., <Maß$_n$>) sowie deren Anordnung kann durch Konfigurationsdateien gesteuert werden. Um die zwischen zwei Achsen entstehenden Flächen sinnvoll interpretieren zu können, ist eine Gruppierung entlang bestimmter Qualitätskriterien (z.B. Verständlichkeit) oder Produktmodellaspekte (z.B. Größe) sinnvoll (vgl. Kapitel 2 und Kapitel 3.2). Alle Meßwerte werden innerhalb des Kiviat-Diagramms so angetragen, daß im Mittelpunkt der minimale (l_j) und auf dem Außenkreis der maximale Meßwert (m_j) aufgetragen wird; der jeweils aktuelle Wert einer Entität (c_j) wird zwischen diesen beiden Extrema entsprechend skaliert.

- *Übersichtorientiert*: Für jedes angewendete Maß besteht die Möglichkeit, eine Übersicht über die entsprechenden Meßwerte innerhalb des betrachteten Systems zu erhalten. Für diesen Zweck werden bei Bedarf Verteilungsdiagramme erstellt (vgl. Kapitel 3.4), deren Parameter wie folgt gesetzt sind (vgl. Abbildung 54):

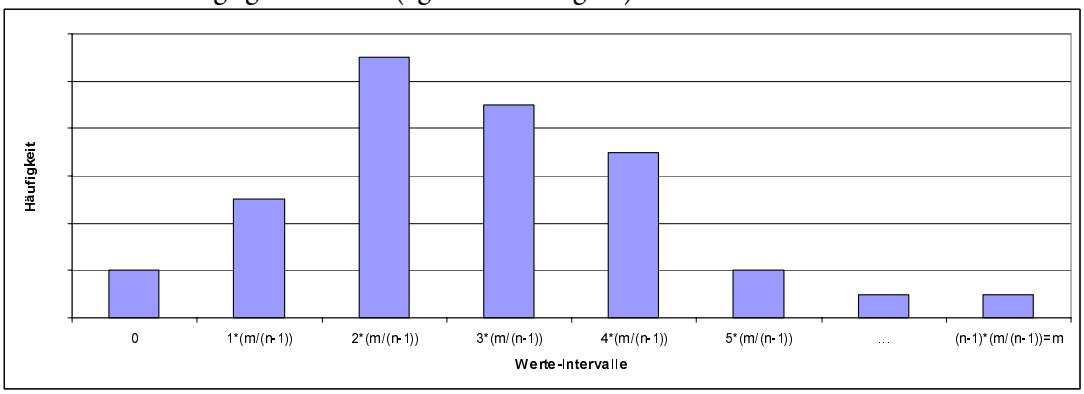


Abbildung 54: Schema eines Verteilungsdiagramms

Auf der x-Achse sind n gleich große Werteintervalle zur Erfassung des Meßwertebereichs, der von 0 bis m reicht, für das Maß aufgetragen: Der i-te y-Wert eines jeden Intervalls gibt an, wieviele Entitäten einen Meßwert besitzen, der im Intervall $]i*(m/(n-1)), (i+1)(m/(n-1))]$ liegt. Das erste Intervall hat hierbei eine Sonderstellung, da nur Entitäten mit einem Meßwert von 0 betrachtet werden. Auf diese Weise ist eine schnelle Übersicht möglich, welche Meßwerte für ein Maß innerhalb des betrachteten Systems wie häufig vorkommen.

- *Strukturdaten*: Neben den Meßdaten können innerhalb von CrocoCosmos auch konkrete Strukturdaten in die Kohäsionsdatensicht eingeblendet werden. Dies umfaßt die mittels Crocodile aus dem Quelltext extrahierten Relationen „Methode benutzt Methode“, „Methode benutzt Attribut“ und „Klasse erbt von Klasse“ (vgl. Kapitel 3.5) und alle darauf aufbauenden Relationen auf übergeordneten Abstraktionsniveaus (z.B. Subsystem erbt von Subsystem, vgl. Kapitel 5.3.2). Diese Daten können wie bereits bei den knotenorientierten Meßdaten entitätorientiert, d.h. bestimmte Relationen für eine bestimmte Entität, und übersichtorientiert, d.h. eine bestimmte Relationsart für alle Entitäten, eingeblendet werden.

CrocoCosmos besitzt als primäres Darstellungsfenster eine dreidimensionale Welt, innerhalb derer die Entitätsübersicht zusammen mit der Entitätserscheinung integriert ist. Ein wesentliches Problem der Exploration umfangreicher Daten und deren dreidimensionaler Visualisierung ist einerseits die Schwierigkeit, eine Übersicht über das System als Ganzes zu erhalten, und andererseits, gezielt Teile des Systems detailliert betrachten und analysieren zu können. Eine Lösung dieses Problems, die innerhalb von CrocoCosmos realisiert ist, liegt in der dynamischen Explorierbarkeit derartiger dreidimensionaler Welten; hierbei sind vor allen Dingen zwei wesentliche Techniken relevant:

- *Abstraktionsniveauselektion*: Die wesentliche Eigenschaft von Multisichten, das System auf unterschiedlichen Abstraktionsniveaus betrachten zu können (vgl. Kapitel 5.3.2), wird dynamisch angeboten, d.h. der Benutzer kann während der Exploration zwischen den verschiedenen Stufen wechseln. Dies geschieht wie bei den knotenorientierten Meßdaten und den Strukturdaten wiederum über zwei unterschiedliche Kontexte:
  - *Entitätorientiert*: Für jede innerhalb des virtuellen Informationsraums dargestellte Entität kann explizit das Abstraktionsniveau geändert werden. Entsprechend der aus CrocoBrowse bekannten hierarchischen Enthaltenseinsrelation (vgl. Kapitel 5.3.2) bedeutet dies das Ausklappen, d.h. das Anzeigen aller enthaltenen Elemente einer angewählten Entität, bzw. das Einklappen, d.h. das Ersetzen aller Entitäten eines gemeinsamen, eine Ebene höher liegenden Container-Elements durch eben diesen Container.
  - *Übersichtorientiert*: Das System kann global alle Elemente einer auszuwählenden Hierarchiestufe darstellen.
- *Navigation*: Ein häufiges Problem größerer Visualisierungen ist das sogenannte *Fokus-versus-Kontext-Problem* ([Chen99], S. 68f), d.h. auf der einen Seite die Notwendigkeit, alle darzustellenden Objekte gleichzeitig zu betrachten, und auf der anderen Seite die damit verbundene Reduzierung der für ein Objekt zur Verfügung stehenden Darstellungsfläche bzw. des Darstellungsraums. Um der damit entstehenden Gefahr der Diagrammüberladung, deren übergroße Informationsdichte innerhalb einer Visualisierung deren Verständnis erschwert, zu begegnen, sind neben trickreichen Techniken – wie z.B. der von Furnas 1981 vorgestellte *Fisheye-View* ([Furn81]) oder die von Munzner erforschten und prototypisch implementierten *hyperbolischen Räume* [Munz98] – besonders Möglichkeiten zur Navigation innerhalb der Welt hilfreich, d.h. das Wechseln des Betrachtungswinkels und des Betrachtungsstandpunkts. Standardnavigationstechniken innerhalb einer dreidimensionalen Welt mit einer Höhe (Y-Wert), einer Breite (X-Wert) und einer Tiefe (Z-Wert) sind z.B. (vgl. Kapitel 1.4 in [Schl98]):
  - *Walk*, d.h. das Verändern des Betrachtungsstandpunkts innerhalb der Welt mit einem fixen Y-Wert von 0 und veränderlichen X- und Z-Werten. Diese Navigation entspricht dem Durchschreiten der Welt auf einer Ebene.
  - *Pan*, d.h. das Verändern des Betrachtungsstandpunkts innerhalb der Welt mit einem fixen Z-Wert und veränderlichem X- und Y-Wert. Diese Navigation entspricht den üblichen Scrollbars fensterorientierter Systeme.
  - *Slide*, d.h. das Verändern des Betrachtungsstandpunkts innerhalb der Welt ausgehend von einem fixen Start-Y-Wert und veränderlichen X- und Z-Werten. Diese Navigation entspricht dem Durchfliegen der Welt.
  - *Turn*, d.h. das Verändern des Betrachtungswinkels innerhalb der Welt entlang aller drei Koordinaten. Häufig wird hierbei ein zu bestimmender Punkt als Drehpunkt verwendet, d.h. die Veränderung der X-, Y- und Z-Werte geschieht um diesen Fixpunkt, dessen Koordinaten während der Rotation auf dem Bildschirm fix bleiben.

  Alle diese Navigationstechniken werden von CrocoCosmos durch die Verwendung eines *VRML*-Clients realisiert: Die *Virtual Reality Modelling Language* bietet Sprachelemente zur Beschreibung dreidimensionaler Strukturen, die anschließend mittels handelsüblicher VRML-Clients dargestellt und unter Verwendung oben genannter Navigationstechniken explorierbar sind (vgl. [Schl98], [DäPa98]). Zur Verstärkung der Tiefeninformationen unterstützen diese Programme zusätzlich sogenannte *Shutter-Brillen*, mit deren Hilfe es möglich ist, zwei jeweils mit halber Monitortaktfrequenz abwechselnd dargestellte Bilder durch gezieltes Verdecken des „falschen Bildes“ dem jeweiligen Auge zuzuordnen und somit echte Tiefeninformationen zu übertragen. Diese Technik wurde ebenfalls für die Durchführung der Beispielprojekte angewendet.

Die Kombination der Abstraktionsniveauselektion und der Navigation birgt ein Problem für die Anordnung der Entitäten entsprechend eines konkreten Distanzmaßes: Da die Instanziierung des generischen Distanzmaßes jeweils nur für Entitäten eines Abstraktionsniveaus definiert ist, existieren zwei grundsätzlich verschiedene Lösungen, gleichzeitig unterschiedliche Abstraktionsniveaus darzustellen:

1. Für jedes Abstraktionsniveau wird ein eigenes generisches Distanzmaß instanziiert und für das zu vermessende System angewendet. Für jedes Abstraktionsniveau werden anschließend konkrete, dreidimensionale Positionen der Entitäten errechnet. Bzgl. der beiden Selektionsmechanismen bedeutet dies für die
   - *übersichtorientierte Abstraktionsniveauselektion*: Alle Entitäten des alten Niveaus werden gelöscht und durch die neuen Entitäten mit ihren vollständig neuen Positionen ersetzt.
   - *entitätorientierte Abstraktionsniveauselektion*: Beim Verfeinern der Hierarchiestufe wird die zu ersetzende Entität gelöscht und durch die enthaltenen Entitäten mit ihren vollständig neuen Positionen ersetzt. Beim umgekehrten Vorgehen werden alle Entitäten eines gemeinsamen, eine Ebene höher liegenden Container-Elements durch eben diesen Container mit seiner eigenen, vollständig neuen Position ersetzt.
2. Die Distanzen werden für eine tief liegende Abstraktionsstufe berechnet und entsprechende Positionen ermittelt. Die Positionen der darüber liegenden Elemente lassen sich auffassen als der geometrische Mittelpunkt der in ihnen definierten Entitäten.

Beide Lösungen haben sowohl Vor- als auch Nachteile: Der Hauptvorteil der ersten Variante ist die aufgrund der innerhalb jeder Abstraktionsstufe vorgenommenen, expliziten Distanzmaßdefinition wohldefinierte Möglichkeit der Interpretation der Abstände zwischen den Entitäten. Im Gegensatz zur übersichtorientierten ist dies allerdings für die entitätorientierte Abstraktionsniveauselektion nur noch bedingt möglich, da evtl. unterschiedliche Distanzkonzepte mit jeweils unterschiedlichen Interpretationen gleichzeitig dargestellt werden. Eine ein Abstraktionsniveau überschreitende Interpretation ist dann kaum mehr möglich. Ein weiteres wesentliches Problem dieser Lösung ist der Orientierungsverlust innerhalb der Welten: Aufgrund der separaten Positionsbestimmung besteht keinerlei lokaler Zusammenhang zwischen Elementen unterschiedlicher Abstraktionsebenen. Dies wiegt besonders bei Containerentitäten schwer, die nur ein Element besitzen. Eine Orientierung innerhalb der Welt, insbesondere über verschiedene Abstraktionsniveaus hinweg, ist damit kaum möglich. Dieser Nachteil wird durch die zweite Lösung größtenteils kompensiert: Speziell bei Containerentitäten, die nur ein Element besitzen, ist die Position identisch, d.h. ein Verfolgen konkreter Entitäten während des Explorierens wird unterstützt.
In vielen praktischen Versuchen dominierte dieser Vorteil über die damit verbundene Schwäche der Interpretation der Distanzen auf übergeordneten Abstraktionsniveaus: Die Nähe zweier Entitäten läßt die Vermutung zu, daß die in ihnen enthaltenen Elemente ebenfalls eine große Nähe besitzen. Dies ist allerdings kein hinreichendes Kriterium sondern muß durch Verfeinerung der Sichten validiert werden. Innerhalb der Projekte wurden aufgrund der besseren Orientierung die Distanzen für Klassen (Abschnitt 9.2 und 9.3.1) bzw. Methoden und Attribute (Abschnitt 9.3.2) berechnet und alle Positionen der Entitäten darüber liegender Abstraktionsniveaus durch die Variante 2 bestimmt.

Ein typischer, mit Annotationen angereicherter Screenshot von CrocoCosmos ist in Abbildung 55 dargestellt. Die Ziffern der Annotationen beziehen sich dabei auf folgende, oben erläuterte Funktionalität:

1.x Kohäsionsdaten: Dargestellt wird die Entitätsübersicht (1a) zusammen mit der Entitätsdarstellung (1b). Die Parameter für die letzteren Daten werden für die jeweilige Abstraktionsstufe (von oben nach unten: Subsystem, Datei, Klasse) unter Verwendung eines Skalierungsfaktors in (1c) eingestellt.

2.x Strukturdaten: Mittels des Pull-Down-Menüs (2a) kann zwischen verschiedenen Modi gewählt werden, die jeweils unterschiedliches Verhalten für die Selektion einer Entität repräsentieren (s.u.). Diese umfassen u.a. die explizite Darstellung von Strukturdaten wie z.B. Vererbungsbeziehungen (2b) und nähere Angaben zur Identifikation (2c).

3.x Navigation: Die Navigationstechnik Walk wird durch (3a), Pan durch (3b), Slide (hier als Fly bezeichnet) durch (3c) und Turn durch (3d) angeboten.

4.x Die zusätzlich mittels (4a) darzustellende Projekthierarchie (vgl. Abbildung 52) kann durch Auswahl von (4b) für die Navigation innerhalb der Welt verwendet werden.

5.x Abstraktionsniveauselektion: Das Pull-Down-Menü (2a) beinhaltet ebenfalls die zwei beschriebenen Selektionsmechanismen. In Abbildung 55 sind unterschiedliche Abstraktionsniveaus durch unterschiedliche Formen symbolisiert (vgl. 5a und 1b).

Das Pull-Down-Menü ermöglicht darüber hinaus die Generierung aller oben beschriebenen Darstellungen knotenorientierter Meßdaten.

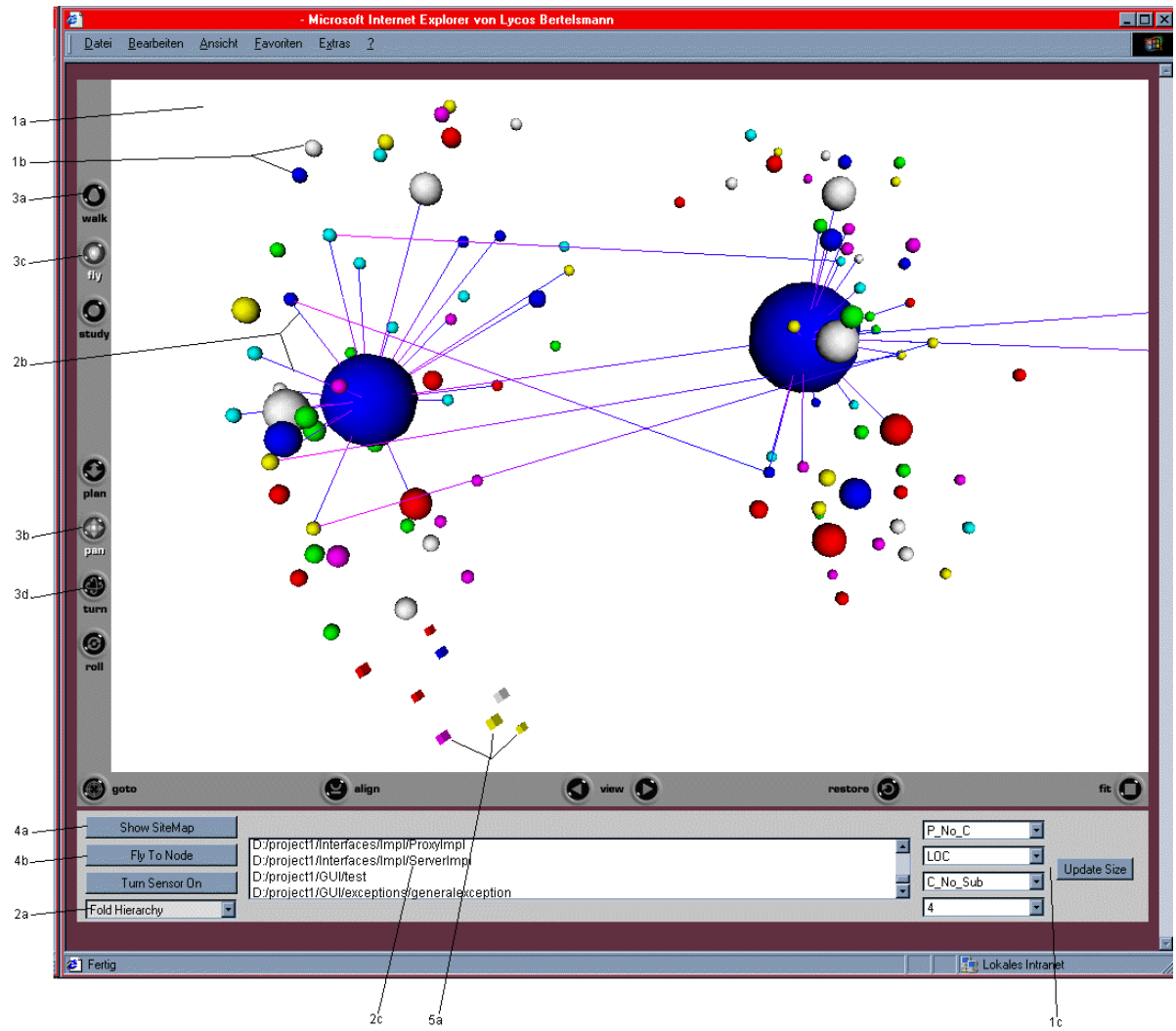


Abbildung 55[*]: Annotierter Beispielscreenshot von CrocoCosmos

9.2 Externe Validierung innerhalb des Prozesses „Beurteilung von Produktqualität“

Die wesentliche externe Validierung soll in dieser Arbeit für zwei typische Projekte demonstriert werden, in denen real eingesetzte Softwareprodukte bzgl. ihrer Qualität beurteilt werden sollten. Um den von Schneidewind geforderten Mindestkriterien einer externen Validierung zu genügen (vgl. Kapitel 4.3.2), wird für den Prozeß der Meßanwendung und der Auswertung der Ergebnisse eine Rückkopplungsschleife (vgl. Feedback Loop in Kapitel 3) für einen zweiten Prozeßdurchlauf verwendet, um zufällige Korrelationen auszuschließen und der Notwendigkeit der Individualisierung des zugrundeliegenden Qualitätsmodells gerecht zu werden. Der Prozeß, der bei beiden Projekten angewendet wurde, kann wie in Abbildung 56 dargestellt werden und entspricht im wesentlichen dem in [SiRuKö98] beschriebenen Prozeß zur Validierung der meßwertbasierten Review-Vorbereitung:

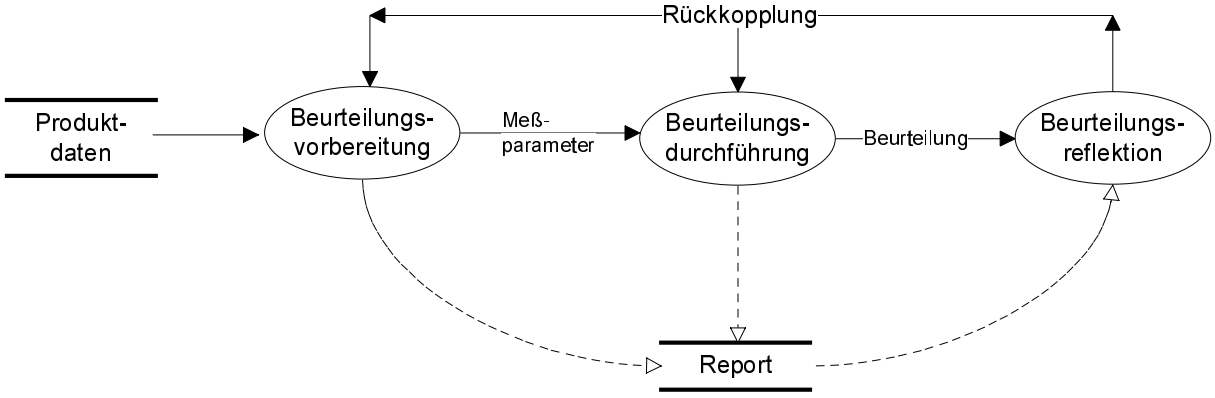


Abbildung 56: Prozeß zur Validierung der meßwertbasierten Qualitätseinschätzung

Die einzelnen Prozeßschritte wurden jeweils wie folgt verfeinert:

1 *Beurteilungsvorbereitung*:

1.1 *Vorbereitung der Produktdaten*: Da in beiden Projekten als ausschließliche Produktdaten der Quelltext selbst vorhanden war, beschränkte sich dieser Teilprozeß auf dessen Verfügbarmachung und Einbindung in die Meßumgebung Crocodile (vgl. Kapitel 3.5). Jeweils im zweiten Prozeßdurchlauf fand darüber hinaus eine detaillierte, durch Diskussion entstandene Komponentenfilterung (vgl. Kapitel 3.5) und Vererbungskontextsetzung (vgl. Kapitel 8.2) statt.

1.2 *Vorbereitung der Meßumgebung*: Dieser Teilprozeß beschränkte sich im wesentlichen auf die Erstellung eines angepaßten Qualitätsmodells. In beiden Projekten wurde für den ersten Projektdurchlauf jeweils ein initiales Qualitätsmodell verwendet: Unter Verwendung der GQM-Methode zur Dekomposition betrachtenswerter Aspekte bis auf die Ebene von Maßen und unter Verwendung des MQG-Vorgehens zur Integration bisheriger sinnvoll erscheinender Maße ist ein graphartiges Qualitätsmodell entstanden (vgl. Kapitel 2.3). Die betrachtenswerten Aspekte innerhalb dieses initialen Qualitätsmodells sind vor allen Dingen
- eine bzgl. unterschiedlicher Größenabstraktionen ausgewogene Größe,
- eine bzgl. unterschiedlicher Kopplungsabstraktionen moderate Kopplung und
- eine bzgl. der unterschiedlichen vorgestellten Kohäsionskonzepte gute Kohäsion.
Nach dieser Verfeinerung des Qualitätsbegriffs in verschiedene Qualitätsaspekte wurden letztere verfeinert in die verschiedenen Abstraktionsebenen, die bzgl. der identifizierten Qualitätsaspekte zu untersuchen waren (z.B. Klassen, Subsysteme etc.).

Die darauf aufbauenden Verfeinerungen, die Einfluß auf den Qualitätsaspekt auf der jeweiligen Abstraktionsebene haben, mündeten in die verschiedenen Maße. Ein Beispiel für einen Kantenpfad eines derart in ein Qualitätsmodell integrierten Maßes ist z.B. Qualität→ Kopplung→ Klassen→ Interaktionskopplung→ (Anzahl von Klassen, zu denen eine Klasse Interaktionskopplungen besitzt) (s.u. Maß "C_EffIntCp_C"). Neben der in Kapitel 2 und Kapitel 3 geforderten, sorgfältigen Maßdefinition wurde jedes Maß bzgl. der Werteauswirkungen umgangssprachlich beschrieben. Dies umfaßt ebenfalls Abhängigkeiten zu anderen Meßwerten, die innerhalb von CrocoCosmos einfach zu identifizieren sind (z.B. gleichzeitige Betrachtung der Anzahl unterschiedlicher Subsysteme, aus denen die benutzten Klassen stammen).

Erst im jeweils zweiten Durchlauf durch den Prozeß (und nach entsprechender Diskussion mit dem jeweiligen Projektpartner) wurden Teilaspekte des Qualitätsmodells erweitert, modifiziert oder entfernt.

1.3 *Planung der Produktbeurteilung*: Um den Anforderungen der Softwaretechnik nach einem systematischen und disziplinierten Vorgehen gerecht zu werden (vgl. Kapitel 3) und um eine spätere Effizienzbeurteilung der gewählten Technik zu unterstützen, ist der gesamte Folgeprozeß als präzises Projekt geplant, d.h. sowohl materielle, personelle als auch zeitliche Ressourcen wurden festgelegt. In beiden Fällen wurden als Rahmenparameter festgelegt:

- Zwei Windows®-Rechner (einer für die Vermessung sowie die Visualisierung und einer für die Report-Erstellung mittels Standard-Office-Werkzeugen),
- zwei Personen mit zusammen ca. 25 Personenstunden und
- 14 Tage als zeitlicher Rahmen zwischen Produktdateneingang und fertiger Qualitätsbeurteilung in Form eines fertiggestellten Reports (s.u.).

2 *Beurteilungsdurchführung*:

2.1 *Meßdurchführung*: Dieser Teilprozeß benötigt im wesentlichen Rechenleistung; der Anstoß der Vermessung und das Festlegen von Parametern wie z.B. gewünschte Kohäsionsart oder Selektion des erstellten Qualitätsmodells geschehen interaktiv und oberflächengestützt im Vorfeld. Die Vermessung selbst benötigt ungefähr für Systeme mit etwas über 1000 Klassen und den unten beschriebenen Produktmaßen zwei Stunden.

2.2 *Meßvisualisierung*: In diesem Teilprozeß werden die in der Meßdurchführung errechneten Meßwerte visualisiert. Während dies für knotenorientierte Maße in Form von Übersichtsdiagrammen keinerlei vorbereitende Tätigkeiten benötigt und erst auf Verlangen erstellt werden kann (vgl. Abschnitt 9.1), müssen die ermittelten Distanzen auf Positionen innerhalb des dreidimensionalen Raums überführt werden (vgl. Kapitel 6.4.2). Dies muß für jedes gewünschte Kohäsionskonzept und die damit verbundene Distanzmatrix separat durchgeführt werden. Für den verwendeten Spring-Embedder als nichtlineare Optimierungstechnik kann keine fixe Mindestzeit angegeben werden. Innerhalb beider Projekte wurde der Spring-Embedder jeweils mindestens zwei Stunden auf einem normalen PC für ein Distanzkonzept angewendet. Die von CrocoCosmos angebotene Visualisierung, die die errechneten Positionen verwendet, kann dagegen wieder auf Verlangen erstellt werden.

2.3 *Meßwertinterpretation*: Dieser Teilprozeß ist der arbeitsintensivste Schritt und stellt den Kern der Qualitätseinschätzung dar. Er besteht aus der Exploration der von CrocoCosmos angebotenen Visualisierungen und deren Interpretation entlang des verwendeten Qualitätsmodells. Die Exploration selbst, die, wenn konkrete negative oder positive Befunde identifiziert wurden, den Report in Listenform anfüllt, kann grob durch die Identifikation folgender Charakteristika umschrieben werden:

- *Extremwertanalyse*, d.h. das Finden von den bzgl. jeweils eines Maßes kleinsten und größten Entitäten: Diese innerhalb der dreidimensionalen Visualisierung i.d.R. sehr schnell zu identifizierenden Entitäten stellen einen guten Ausgangspunkt für die Exploration dar, da diese Entitäten allein aufgrund des einen Meßwerts *besonders* sind. In einer daran anschließenden detaillierteren Analyse kann der Grad der Besonderheit entlang des Verteilungsdiagramms untersucht und mittels des Kiviat-Diagramms jeweils das Zusammenspiel der verschiedenen Meßwerte der unterschiedlichen Maße betrachtet werden. Der aufgrund der Kohäsionsdarstellung direkt ersichtliche Kontext und das darin mögliche interaktive Einblenden von zusätzlichen Strukturdaten sowie das Verfolgen der jeweiligen Entität über verschiedene Abstraktionsstufen hinweg ermöglicht ein kontextbezogenes Reverse-Engineering (vgl. Kapitel 3.3.3), das für die Interpretation sehr hilfreich ist.
- *Cluster-Identifikation*, d.h. das Finden von Entitätsgruppen entsprechend des verwendeten Kohäsionskonzepts: Ein besonders wichtiges und in der Visualisierung sehr zügig erkennbares Charakteristikum ist die Bildung von Entitätsgruppen, da diese entsprechend des zugrundeliegenden Kohäsionskonzepts stark zusammengehören. Innerhalb der interaktionbasierten Kohäsion kann ein gutes Subsystem sich z.B. dadurch auszeichnen, daß ein Großteil der Klassen des Subsystems, d.h. alle Klassen mit derselben Färbung, eng beieinander liegen, aber gleichzeitig eine große Entfernung zu den anderen Systemteilen besitzen. Lediglich eine geringe Anzahl von *Fassadenklassen* ([GHJV96], S. 189ff) treten lokal als Bindeglied zwischen dem Subsystem und anderen Systemteilen hervor und liegen zwischen dem Subsystem und den es benutzenden *Client-Klassen* ([ebd.]). Genauso ist es in der vererbungbasierten Kohäsion leicht möglich, Vererbungsteilbäume zu identifizieren, die zusammen mit klassischen knotenorientierten Maßen eine vertiefte Analyse der Vererbungsstruktur ermöglichen.
- *Anticluster-Identifikation*, d.h. das Finden von Entitätsgruppen-Anomalien entsprechend des verwendeten Kohäsionskonzepts: Ein ebenfalls wichtiges und wiederum in der Visualisierung sehr zügig erkennbares Charakteristikum ist die Bildung von separierten Teilen, da diese entsprechend des zugrundeliegenden Kohäsionskonzepts sich stark abstoßen. Innerhalb der interaktionbasierten Kohäsion fallen *tote Klassen* z.B. durch eine große Distanz zu allen anderen Elementen auf; unter der Vorgabe, daß Klassen eines Subsystems bzgl. der Interaktion eng zusammenarbeiten sollen, fallen ungünstig gebildete Subsysteme dadurch auf, daß ihre Klassen überall im System verteilt sind. Auch derartige visuelle Phänomene können unter Einbeziehung konkreter Strukturdaten und weiterer Meßwerte zügig interpretiert und ggf. in den Report mit übernommen werden.

Die derartige Exploration eines neuen Systems hat in beiden Projekten annähernd 15 Personenstunden benötigt. Das bewußte Begrenzen der Zeit dieses Teilprozesses ist notwendig, um die Vorteile eines meßwertbasierten Vorgehens zu erhalten. Je mehr Zeit hierfür notwendig ist, desto früher kann darüber nachgedacht werden, neben der Meßwertvisualisierung auch andere, deutlich zeitintensivere Techniken wie z.B. das vollständige Reviewing (vgl. Kapitel 3.3.2) anzuwenden. Innerhalb der verwendeten 15 Stunden ist dies allerdings bei weitem nicht möglich.

Das Hauptergebnis dieses Arbeitsschritts ist der Report, dessen Erstellung jeweils bis zu 8 Stunden benötigt hat. In ihm sind neben der Qualitätseinschätzung auch Angaben zum Vorgehen, das verwendete Qualitätsmodell und Übersichtsdiagramme für einige Maße enthalten. Die Qualitätseinschätzung äußert sich in verschiedenen Eindrücken und Restrukturierungsempfehlungen (im folgenden als Anmerkung zusammengefaßt): Während die Eindrücke lediglich Aussagen über die Qualität beinhalten, umfassen Restrukturierungsempfehlungen zusätzlich mögliche Maßnahmen zur Beseitigung der Qualitätsdefizite. Auch wenn diese zusätzliche Art von Informationen nicht im primären Fokus der Qualitätseinschätzung liegt, so lassen sie sich doch aus der Visualisierung häufig so direkt ablesen, daß dafür kein zusätzlicher Mehraufwand betrieben werden muß. Da die Projekte in beiden Fällen vom Auftraggeber selbst gewartet werden, bestand außerdem begründetes Interesse an derartigen Empfehlungen.

3 *Beurteilungsreflexion*: Speziell für die externe Validierung ist es notwendig, den ausschließlich auf Meßwerten basierenden Report mit der Qualitätseinschätzung von Softwareentwicklern, die das untersuchte System sehr gut kennen, vergleichen zu können. Um sowohl den Aufwand für die Softwareentwickler zu minimieren als auch die Auswertung zu vereinfachen, wurde für den Fragebogen die *Rating-Methode*, die eine spezielle Form der Implementierung *geschlossener Fragen* darstellt, angewendet ([Sinc90]). Hierbei geht es darum, jedes zu beurteilende Objekt bzgl. des zu untersuchenden Kriteriums, in diesem Fall die Zustimmung zu der jeweiligen Anmerkung, ordinal mittels vorgegebener fünf Bewertungsstufen, die im folgenden erläutert werden, einzuschätzen.

Bewertungsstufe	Bedeutung
„Vollständig richtig“	Die Anmerkung entspricht exakt dem Eindruck der Entwickler.
„Größtenteils richtig“	Die Anmerkung entspricht im wesentlichen dem Eindruck der Entwickler.
„Indifferent“	Die Anmerkung kann entweder nicht beurteilt werden oder besitzt sowohl zustimmungswürdige als auch ablehnungswürdige Aspekte.
„Größtenteils falsch“	Die Anmerkung widerspricht in großen Teilen dem Eindruck der Entwickler.
„Vollständig falsch“	Die Anmerkung widerspricht vollständig dem Eindruck der Entwickler.

Bzgl. der als vollständig oder größtenteils richtig beurteilten Anmerkungen ist darüber hinaus ihr jeweiliger Neuigkeitsgehalt interessant, d.h. zu welchem Grad ist der Inhalt einer Anmerkung bereits im Vorfeld der Beurteilung bekannt. Für diesen Zweck werden die entsprechenden positiv bewerteten Anmerkungen zusätzlich bzgl. folgender drei Bewertungsstufen für den Neuigkeitsgehalt einer Anmerkung eingeschätzt:

Bewertungsstufe	Bedeutung
„Vollständig neu“	Die Anmerkung entspricht dem Eindruck der Entwickler, ist ihnen aber bis zur Beurteilung nicht bekannt gewesen. Derartige Anmerkungen sind für die Entwickler sehr wertvoll, da sie dadurch neue Einblicke in das System erhalten.
„Bestätigung ungenauen Wissens“	Die Anmerkung entspricht dem Eindruck der Entwickler und bestätigt ein für sie bis dahin nur sehr ungenaues Wissen um das System. In direkten Gesprächen wird diese Art des Systemwissens häufig als „Gefühl“ oder „Ahnung“ beschrieben. Derartige Anmerkungen sind für die Entwickler wertvoll, da sie durch die Beurteilung eine durch Meßwerte objektivierte Basis für ihre bis dahin nur unscharf belegbaren Eindrücke erhalten und damit in der Lage sind, diese zu verfeinern oder/und zu modifizieren.
„Gut bekannt“	Die Anmerkung entspricht dem Eindruck der Entwickler, war ihnen aber bereits im Vorfeld bekannt. Derartige Anmerkungen sind primär für die externe Validierung wichtig, da sie – wie auch die beiden oberen Bewertungsstufen – belegen, daß eine meßwertbasierte Analyse dem Wissen kompetenter Entwickler entsprechende Aussagen ergeben; der Wert solcher Anmerkungen für die Entwickler liegt eher im „Explizieren“ derartigen Wissens in Form des Reports.

Der beantwortete Fragebogen sowie dessen gemeinsame Auswertung wurden für die Anpassung des Qualitätsmodells verwendet. Die konkreten Modifikationen und deren Motive sind jeweils im zweiten Prozeßdurchlauf der Projekte beschrieben.

9.2.1 Prozeßdurchlauf für die JWAM-Version 1.4

Unter Leitung von Heinz Züllighoven hat der Arbeitsbereich *Softwaretechnik* der Universität Hamburg seit 1991 objektorientierte Anwendungsentwicklung nach dem *Werkzeug-Automat-Material*-Ansatz (*WAM*) durchgeführt und evaluiert. Dieser stellt eine besondere Sicht auf objektorientierte Softwareentwicklung zur Verfügung und bietet spezielle Techniken für die Konstruktion, Analyse und Dokumentation. Zentral für den WAM-Ansatz sind die drei Aspekte Werkzeug, Automat und Material: *Werkzeuge* spiegeln die Organisation der Arbeit (als Handlung) wider, d.h. sie repräsentieren wiederkehrende Arbeitsabläufe, die bei unterschiedlichen Aufgaben und Zielsetzungen nützlich sind und von Benutzern verwendet werden, um die Arbeitsobjekte in Form der Materialien zu manipulieren. Die *Materialien* selbst spiegeln die Arbeitsgegenstände wider, die schließlich Teil (oder Ganzes) von Arbeitsergebnissen und -zwischenergebnissen werden und die niemals direkt benutzt werden können. Der dritte wichtige Aspekt sind die *Automaten*, die im Vorfeld vollständig festgelegte Aufgaben automatisch und ohne Eingriff von außen erledigen. Eine detailliertere Beschreibung des WAM-Ansatzes ist z.B. in [Züll98] zu finden.
Ende 1996 wurde innerhalb der Gruppe damit begonnen, ein JAVA-Framework für die effizientere Erstellung von Software nach dem WAM-Ansatz zu entwickeln: JWAM. 1999 wurde JWAM in die neu gegründete *Apcon Workplace Solutions (WPS) GmbH* mit übernommen und wird dort seit 2000 für die kommerzielle Softwareerstellung verwendet und intensiv weiterentwickelt. Für die in dieser Arbeit beschriebene externe Validierung wurden die beiden JWAM-Versionen 1.4 und 1.5 verwendet. Die Beschreibung der Qualitätsbeurteilung der JWAM-Version 1.4 geschieht im folgenden entlang des in Abbildung 56 dargestellten Prozesses.

Beurteilungsvorbereitung
Da der Quelltext von JWAM als Public-Domain-Software vertrieben wird, konnten die Produktdaten für die Qualitätsbeurteilung vom offiziellen JWAM-Software-Server bezogen werden. Diese Daten waren für den ersten Prozeßdurchlauf zugleich die einzigen Daten.
Bereits während der Produktmodellerstellung mittels Crocodile sind folgende Systemgrößen der JWAM-Version 1.4 identifiziert worden:

- 100 Subsysteme mit zusammen 922 Dateien,
- 1287 Klassen mit zusammen 7818 Methoden und 2235 Attributen,
- 655 Vererbungsrelationen,
- 11039 Benutztbeziehungen zwischen Methoden und 5656 Benutztbeziehungen zwischen Methoden und Attributen.

Aufgrund des Fehlens weiterer Informationen wurde für den ersten Durchlauf das gesamte Projekt vermessen und beurteilt. Auf die Setzung eines Vererbungskontexts, d.h. das Definieren von Klassen, die entsprechend des Kapitels 8.2 für eine Vermessung mittels der Flatten-Technik vorbereitet werden müssen, wurde ebenfalls verzichtet.
Für den ersten Prozeßdurchlauf wurde ein Standard-Qualitätsmodell für große JAVA-Systeme verwendet, dessen Einträge und Abhängigkeiten zu Qualitätskriterien auf eigenen Erfahrungen und einer Literaturrecherche beruhen. Die insgesamt 18 dort definierten Softwaremaße können entsprechend der Klassifizierung von Produktmodellen (vgl. Kapitel 3.2.ff) wie folgt gegliedert werden:

- Größenmaße: Für die verschiedenen Abstraktionsniveaus objektorientierter Systeme (vgl. Kapitel 8.1) wurden verschiedene Größenmodellmaße angewendet (vgl. Kapitel 3.2.1). Sie alle umfassen quantitative Angaben zu einer Entität über die Anzahl in ihr enthaltener Entitäten einer darunter liegenden Abstraktionsschicht. Folgende 9 Größenmaße wurden verwendet:

Abstraktions-niveau	Bezeichnung[29]	Beschreibung
Sub-system	*P_Cl*	Das Maß *P_Cl* bestimmt für ein Subsystem die Anzahl der darin deklarierten Klassen. Mitgezählt werden Interfaces, abstrakte Klassen sowie nested classes.
Datei	*F_BLoC*	Das Maß *F_BLoC* bestimmt für eine Datei die Anzahl existierender Carriage-Returns, d.h. es wird der Brutto-LoC-Wert ermittelt.
	F_Del	Das Maß *F_Del* bestimmt für eine Datei die Anzahl existierender Befehlstrennungen („ ; “). Sie dient als Approximation der innerhalb einer Datei vorkommenden JAVA-Befehle.
	F_CyclN	Das Maß *F_CyclN* bestimmt für eine Datei die Anzahl kontrollflußmodifizierender Anweisungen (`if`, `switch`, `while`, etc.). Sie dient als Approximation der zyklomatischen Zahl von McCabe.
	F_Comm	Das Maß *F_Comm* bestimmt für eine Datei die Anzahl von Kommentarzeilen, die jeweils durch „//“ eingeleitet werden.
Klasse	*C_PubM*	Das Maß *C_PubM* bestimmt für eine Klasse, wieviele öffentliche Methoden sie deklariert oder implementiert. Nicht betrachtet werden geerbte Methoden und nicht überschriebene Standardkonstruktoren und Standard-destruktoren.
	C_PubA	Das Maß *C_PubA* bestimmt für eine Klasse die Anzahl öffentlich deklarierter Attribute. Es werden ebenfalls die in JAVA als `const` Ersatz üblichen `static final` Attribute mitgezählt. Nicht betrachtet werden geerbte Attribute.
	C_WMCoCh	Das Maß *C_WMCoCh* bestimmt für eine Klasse die für die Implementierung ihrer Methoden notwendigen Zeichen. Gezählt werden alle Zeichen (inkl. Kommentare innerhalb der Methoden). Nicht betrachtet werden geerbte, nicht überschriebene Methoden.
	C_DovM	Das Maß *C_DovM* bestimmt für eine Klasse die Anzahl derjenigen Methoden, die direkt geerbte Methoden überschreiben. Eine Klasse, die eine Schnittstelle implementiert, überschreibt in diesem Sinne alle Methoden der Schnittstelle.

- Kopplungsmaße: Für die verschiedenen Abstraktionsniveaus objektorientierter Systeme (vgl. Kapitel 8.1) wurden verschiedene Kopplungsmodellmaße angewendet (vgl. Kapitel 3.2.2). Sie alle umfassen quantitative Angaben zu einer Entität, die den Grad der Vernetzung zu anderen Entitäten widerspiegeln. Als Kopplung wurden hier die Interaktionskopplung (vgl. Kapitel 7.2.1) und die Vererbungskopplung berücksichtigt (vgl. Kapitel 7.2.3), die jeweils auf alle

[29] Die Bezeichnungen für die Größenmaße sind einheitlich nach folgendem Muster aufgebaut: <Betrachtete Ebene>_<Gezählte Elemente>. C_PubM ist folglich ein Klassenmaß, das die Anzahl öffentlicher Methoden zählt.

höher liegenden Abstraktionsschichten hochaggregiert wurden (vgl. Abschnitt 9.1). Folgende 9 Kopplungsmaße wurden verwendet:

Abstraktions-niveau	Bezeichnung[30]	Beschreibung
Subsystem	*P_EffIntCp_P*	Das Maß *P_EffIntCp_P* bestimmt für ein Subsystem die Anzahl anderer Subsysteme, zu denen Interaktionen bestehen[31]. Als Interaktion wird ein Methodenaufruf betrachtet. Polymorphe Strukturen werden als Interaktion zur Superklasse abgebildet.
	P_AffIntCp_P	Das Maß *P_AffIntCp_P* bestimmt für ein Subsystem die Anzahl anderer Subsysteme, die zum betrachteten Subsystem Interaktionen (s. P_EffIntCp_P) besitzen.
Klasse	*C_Sup_C*	Das Maß *C_Sup_C* bestimmt für eine Klasse die Anzahl direkter Superklassen. Alle Formen der Vererbung werden gleichwertig betrachtet (vgl. Kapitel 8.2).
	C_EffIntCp_C	Das Maß *C_EffIntCp_C* bestimmt für eine Klasse, zu wieviel anderen Klassen Interaktionen (s. *P_EffIntCp_P*) bestehen.
	C_AffIntCp_C	Das Maß *C_AffIntCp_C* bestimmt für eine Klasse die Anzahl anderer Klassen, die zur betrachteten Klasse Interaktionen (s. P_EffIntCp_P) besitzen.
Methode/Attribut	*M_AffIntCp_M*	Das Maß *M_AffIntCp_M* bestimmt für eine Methode einer Klasse die Anzahl von Methoden/Attributen anderer Klassen, zu denen Interaktionen (s. P_EffIntCp_P) bestehen.
	M_AffIntCp_M	Das Maß *M_AffIntCp_M* bestimmt für eine Methode die Anzahl von Methoden anderer Klassen, die zur betrachteten Methode Interaktionen (s. P_EffIntCp_P) besitzen.
	A_AffIntCp_OM	Das Maß *A_AffIntCp_OM* bestimmt für jedes Attribut einer Klasse die Anzahl von Methoden derselben Klasse, die das betrachtete Attribut benutzen.
	A_AffIntCp_FM	Das Maß *A_AffIntCp_FM* bestimmt für jedes Attribut einer Klasse die Anzahl von Methoden fremder Klassen, die das betrachtete Attribut benutzen.

Für die anschließende Visualisierung wurden die notwendigen Daten für die gewichtete, methodenbasierte Interaktionskohäsion (s. Kapitel 8.3.2) und für die allgemeine Vererbungskohäsion (s. Kapitel 8.3.1) berechnet.
Mit dem für alle Prozeßdurchläufe ähnlichen Ressourcenplan (vgl. Abschnitt 9.2) wurde anschließend die Beurteilungsdurchführung begonnen.

[30] Die Bezeichnungen für die Kopplungsmaße sind einheitlich nach folgendem Muster aufgebaut: <Betrachtete Ebene>_<Betrachtete Kopplungsart>_<Ebene des Kopplungspartners>. C_EffIntCp_C ist folglich ein Klassenmaß, das die Anzahl ausgehender Interaktionskopplungen zu anderen Klassen bestimmt.

[31] Um den Begriff der Interaktionskopplung in Anlehnung an Kapitel 7.2.1 hier weiterverwenden zu können, wird diese im folgenden als gerichtet betrachtet, d.h. es gibt einen aktiven Teil, von dem die Interaktionskopplung ausgeht, und einen passiven Teil, von dem etwas verwendet wird.

Beurteilungsdurchführung
Der in diesem Teilprozeß entstandene Report umfaßt 43 Seiten mit 27 Eindrücken und 28 Restrukturierungsempfehlungen. Jede Anmerkung basiert dabei auf einer Vielzahl unterschiedlicher Sichten auf das System, die während der Exploration des dargestellten, virtuellen Informationsraums entstehen. Auch wenn daher eine genaue Zuordnung einer Anmerkung zu speziellen Maßen oder Visualisierungseinstellungen unmöglich ist, wird innerhalb des Reports versucht, jede Anmerkung mit demjenigen Maß zu begründen, das für die Anmerkung *dominant* zu sein scheint. Damit erhält der Report eine einfache, an die jeweils für die Beurteilung verwendeten Maße angepaßte Struktur und erlaubt eine prägnante, anhand konkreter Meßwerte belegbare Präsentation. Beispiele für solche Anmerkungen sind:

Typ	Dominantes Maß	Anmerkung (dem Report entnommen)
Eindruck	P_Cl	„*Die Aufteilung der Klassen auf die Ebene der Subsysteme scheint im wesentlichen sehr sorgfältig vorgenommen worden zu sein. Die Subsystemebene als einer über der Klassenebene liegenden Abstraktion dient damit tatsächlich als zusätzliche Strukturierungsebene und unterstützt das Verständnis des Gesamtsystems.*“
Empfehlung	P_Cl	„*Die 8 Subsysteme* [...], *die jeweils nur eine Klasse enthalten, dienen nicht einer erhöhten Verständlichkeit, sondern lassen das Gesamtsystem aufgrund der künstlich erhöhten Subsystemanzahl komplexer erscheinen. Ein Subsystem mit nur einer Klasse ist damit i.d.R. nicht als zusätzliche Abstraktion verwendbar. Empfehlenswert wäre ein Zusammenfassen mehrerer Subsysteme oder das Einbinden der darin enthaltenen Klassen in bestehende Subsysteme.*“
Eindruck	P_EffIntCp_P	„*Insgesamt sind die Subsysteme sehr kohäsiv, d.h. funktional zusammenhängende Klassen sind auch in einem Subsystem untergebracht.*“
Empfehlung	P_EffIntCp_P	„*Die von den Subsystemen mit den größten P_EffIntCp_P-Werten verwendete Funktionalität ist über sehr viele verschiedene Subsysteme verteilt. Damit ist es nur mit sehr viel Aufwand möglich, diese Subsysteme vollständig zu verstehen, da dies auch eine Kenntnis der jeweils benutzten Subsysteme beinhaltet. Eine Reduktion dieser efferenten Subsystemkopplung durch die Einführung neuer Abstraktionsebenen oder durch das Zusammenlegen kleinerer Subsysteme* [...] *wäre zumindest wünschenswert.*“
Eindruck	C_PubM	„*Fast alle Klassen besitzen eine sehr moderate Anzahl von öffentlich zugänglichen Methoden. Dadurch wird das relativ zügige Einarbeiten in eine einzelne Klasse und in die von ihr angebotene Funktionalität unterstützt. Darüber hinaus läßt diese gute Verteilung eine entsprechend feingranulare Aufteilung der zu implementierenden Funktionen auf Klassen vermuten.*“
Empfehlung	C_PubM	„*Wie bereits bei vielen Größenmaßen weiter oben, fallen wieder die Klassen* [...] *und* [...] *auf. Aufgrund der großen Funktionalität, die von diesen Klassen nach außen angeboten wird, sind sie nur schwer zu verstehen und zu verwenden. Eine Aufteilung in mehrere Teilklassen ist empfehlenswert.*“

Der Report ist zusammen mit dem Auswertungsbogen zwei Wochen nach Eingang der Software an die JWAM-Entwickler geschickt worden.

Beurteilungsreflexion

Mit dem Eingang des ausgefüllten Auswertungsbogens war es möglich, eine erste Validierung des Beurteilungsprozesses und der darin verwendeten Maße vorzunehmen. Das Ergebnis der Bewertungen der Anmerkungen ist in Abbildung 57 dargestellt:

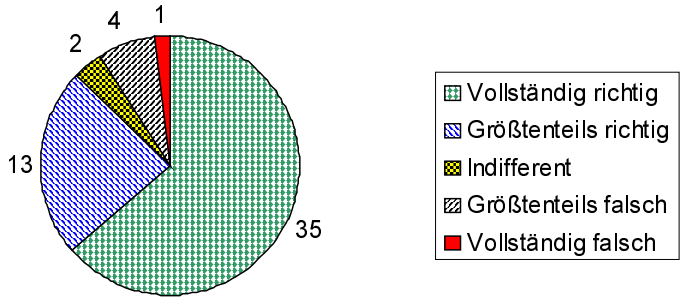


Abbildung 57: Bewertung der Anmerkungen zu JWAM 1.4

Diese sehr guten Ergebnisse, die belegen, daß über 87% der Anmerkungen, die auf einer ausschließlich meßwertbasierten, Ressourcen-minimalen Exploration des Systems beruhen, von den Entwicklern mit ihren detaillierten Kenntnissen des Systems, die weit über die syntaktisch extrahierbare Struktur hinaus gehen, als richtig eingestuft werden, belegen die Validität und Effizienz des gewählten Ansatzes. Dieser positive Eindruck wird durch die Verfeinerung von Anmerkungen in Eindrücke zur ausschließlich qualitativen Beurteilung (vgl. linkes Diagramm in Abbildung 58) und Empfehlungen, die darüber hinaus zusätzlich Restrukturierungsmaßnahmen vorschlagen und damit bzgl. der Bewertung doppelt korrekt sein müssen (vgl. rechtes Diagramm in Abbildung 58), noch weiter untermauert.

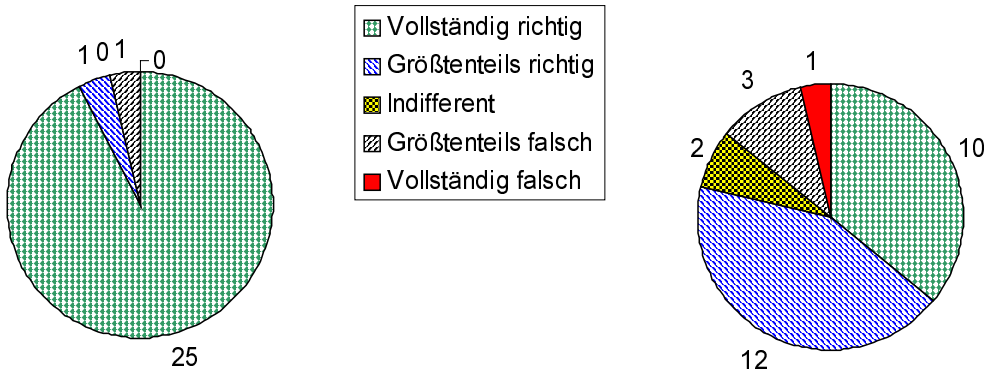


Abbildung 58: Ergebnisse der Auswertung für Anmerkungen (links) und Empfehlungen (rechts)

Die über 96-prozentige Zustimmung der Entwickler zu den Eindrücken sind ein eindrucksvoller Beleg für die erfolgreiche Vorvalidierung des gewählten Ansatzes. Das immer noch gute Ergebnis der Zustimmung von 78% der vorgeschlagenen Restrukturierungsempfehlungen zeigt aber deutlich, daß diese besonders vom jeweiligen Qualitätsverständnis abhängen. Als Beispiel hierfür wird im folgenden die einzige als vollständig falsch beurteilte Restrukturierungsempfehlung diskutiert (s. zweite Anmerkung von den oben aufgezeigten Beispielanmerkungen): Innerhalb aller 8 dort aufgeführten Subsysteme existiert eine jeweils identische, ausschließlich aus einer Klassendeklaration bestehende Klasse. Entsprechend des Standard-Qualitätsmodells mit den Standard-Maßen (vgl. oben) wurden diese Situationen in diesem ersten Durchlauf jeweils als kritisch angemerkt und mit den Restrukturierungsempfehlungen „Zusammenfassen der Subsysteme“ angereichert. Der Grund für die kritische Beurteilung liegt in folgenden verletzten Qualitätskriterien des Standardqualitätsmodells:

- Subsysteme stellen eine Abstraktionsschicht zur Dekomposition des Gesamtproblems in Teilprobleme dar. Für diesen Zweck ist es sinnvoll, jedem Subsystem eine überschaubare, aber zugleich dem Zweck der Dekomposition dienliche Anzahl von Klassen zuzuordnen. Diese Anforderung ist nicht erfüllt für Subsysteme mit einer Klasse.
- Wie die Subsysteme so stellen auch die Klassen eine Abstraktionsschicht zur Dekomposition des Gesamtproblems in Teilprobleme dar. Aus denselben Größenüberlegungen wie bei den Subsystemen folgt wiederum, daß ausschließliche Klassendeklarationen diesem Zweck nicht dienen.

Die totale Ablehnung dieser Restrukturierungsempfehlung liegt in der besonderen Natur der im Vorfeld der Betrachtung nicht genügend berücksichtigten Systemart des Systems, einem wichtigen Einflußfaktor für das Qualitätsmodell (vgl. Kapitel 2.2): Da JWAM ein *Framework* ist, d.h. eine „[...] *fertige Systemarchitektur zur Erstellung von Anwendungen für ein Gebiet*“ ([Kahl98], S.430) darstellt und damit dieses Halbfabrikat vor Ort noch entsprechend konkreter Anforderungen erweitert werden muß, gilt ein besonderer Augenmerk denjenigen Systemstellen, die einen direkten Erweiterungspunkt darstellen. Die vorgefundenen, identischen Klassendeklarationen stellen somit lediglich Platzhalter in Subsystemen dar, die für eine konkrete Instanziierung noch erweitert werden müssen. Die fehlende Berücksichtigung dieses Qualitätskriteriums führte zur vollständig abgelehnten Restrukturierungsempfehlung.
Neben diesen für die eigentliche Validierung notwendigen Daten ist für die Beurteilung des Wertgehalts des Reports für das Unternehmen die Frage nach dem Neuigkeitsgehalt wichtig. Die 48 als größtenteils oder vollständig richtigen Anmerkungen sind diesbezüglich wie in Abbildung 59 dargestellt bewertet worden:

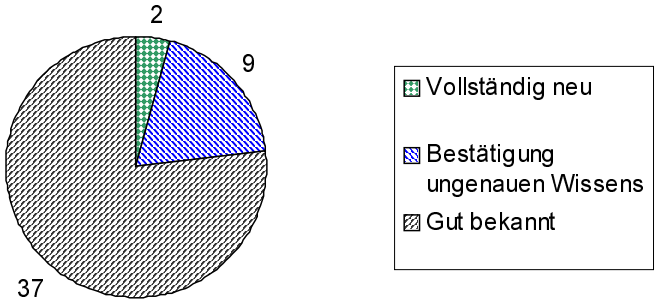


Abbildung 59: Bewertung des Neuigkeitsgehalts der als richtig beurteilten Anmerkungen des Reports

Aus Unternehmenssicht haben vor allen Dingen die 11 Anmerkungen, die entweder ungenaues Wissen festigen oder vollständig neue Einblicke gewähren, Bedeutung. Eine anschließende Diskussion hat aber weniger die angesprochenen Aspekte innerhalb des Reports als Grund des bereits hohen Bekanntheitsgrads identifiziert, sondern vor allen Dingen folgende projektspezifische Parameter:

- Das Gesamtprojekt JWAM ist noch relativ jung, so daß viele grundlegende Entscheidungen und Entwurfsaspekte noch bekannt sind und das Gesetz der zunehmenden Entropie (vgl. Kapitel 8) noch verhältnismäßig geringe Auswirkungen hat.
- Die sogenannte *JWAM-Architecture-Group*, die die Beurteilung vorgenommen hat, besteht größtenteils aus JWAM-Pionieren, die seit Beginn des JWAM-Projekts an dessen Entwicklung beteiligt sind und es damit zwangsläufig sehr gut kennen.
- Ein wesentliches Charakteristikum der JWAM-Entwicklung ist die Anwendung des *eXtreme-Programming* ([Beck00]). Für diesen Kontext wesentlich ist die Forderung, der Quelltext gehöre dem gesamten Projektteam (*Collective Code Ownership*) anstelle einzelner Softwareentwickler (*Individual Code Ownership*). Der damit einhergehenden Forderung, jeder Quelltext müsse von jedem Entwickler des Teams verstanden sein, wird hierbei durch

verschiedene Techniken, wie z.B. intensives Reviewing, Pairprogramming und sehr restriktive Kodierrichtlinien, versucht, gerecht zu werden. Diese Forderung ist für Projektgrößen wie die von JWAM durchaus noch praktikabel.

Für den Prozeß der Validierung ist der Neuigkeitsgehalt sekundär; vielmehr belegt der hohe Bekanntheitsgrad der Anmerkungen das ausgezeichnete fachliche Know-How der JWAM-Architecture-Group und festigt damit die für die Validierung notwendigen Aussagen bzgl. der Richtigkeit der Anmerkungen. Ihre 96-prozentige Zustimmung bzgl. der Eindrücke und ihre 78-prozentige Zustimmung bzgl. der Restrukturierungsempfehlungen belegen damit signifikant, daß der gewählte Ansatz der visuellen Darstellung einer Kombination knoten- und kantenorientierter Maße tatsächlich in der Lage ist, eine dem Messen zugrundeliegende Fragestellung beantworten zu helfen. Die für diesen Kontext notwendigerweise redundant vorhandenen Systemeindrücke – einmal in Form des Reports und einmal durch die JWAM-Entwickler selbst – ist damit in Zukunft nicht mehr notwendig, sondern kann durch ausschließliche Betrachtung des Reports ersetzt werden, da dessen Einschätzungen von Systemkennern bestätigt würden. Dies ist besonders für Projekte wichtig, in denen die Systeminformationen nicht mehr vollständig vorliegen (z.B. aufgrund hoher Personalfluktuation).
Um diese Vorvalidierung zu festigen und zufällige Korrelationen auszuschließen, ist ein weiterer Durchlauf durch den Prozeß notwendig. In ihm wird versucht, die qualitativen Eigenheiten des Projekts besser zu berücksichtigen und den Meßprozeß entsprechend zu modifizieren.

9.2.2 Prozeßdurchlauf für die JWAM-Version 1.5

Einige Wochen nach Auslieferung des Reports für die JWAM-Version 1.4 wurde die neue JWAM-Version 1.5 fertiggestellt. Auch Sie ist entsprechend des Prozesses zur Validierung der meßwertbasierten Qualitätseinschätzung untersucht worden. Die zwei primären Ziele waren:

- Erstellung eines angepaßteren Qualitätsmodells. Hierfür wurden die Auswertungen analysiert (vgl. Abschnitt 9.2.1) und mit dem JWAM-Team darüber hinaus gehende Gespräche geführt.
- Re-Validierung des gewählten Meßansatzes entsprechend des Prozesses zur externen Validierung von Softwaremaßen (vgl. Kapitel 4.3.2).

Beide Punkte stehen teilweise diametral zueinander: Je stärker die Anpassung des Qualitätsmodells und die damit evtl. verbundene neue Ausrichtung des Reports geschieht, desto schwieriger ist die anschließende, beide Prozeßdurchläufe berücksichtigende Validierung, da letztere die Vergleichbarkeit voraussetzt. Die Anpassungen sollten daher so maximal wie möglich sein, ohne allerdings die Möglichkeit der anschließenden Validierung zu behindern.
Eben aus diesen Gründen ist ebenfalls die Möglichkeit einer qualitativen Trendanalyse (vgl. Kapitel 3.3.4), d.h. der gezielten Nachverfolgung von Meßwerten über beide Versionen, um Aussagen über die qualitative Entwicklung des Systems zu machen, nicht primär verfolgt worden, da anderenfalls die Prozeßparameter zu stark hätten verändert werden müssen und damit neue, eventuell abhängige Variablen in das Validierungsexperiment eingebracht worden wären.
Die Beschreibung der Qualitätsbeurteilung von JWAM 1.5 geschieht wie in Abschnitt 9.2.1 entlang des aufgeführten Prozesses aus Abschnitt 9.2.

Beurteilungsvorbereitung

JWAM 1.5 wurde wie die Vorgängerversion vom offiziellen JWAM-Server bezogen. Im Gegensatz zum ersten Prozeßdurchlauf konnte allerdings ein in der Diskussion des ersten Reports erarbeiteter *Meßfokus* gesetzt werden, d.h. einige Subsysteme und Klassen wurden für die weitere Vermessung ausgeblendet (vgl. Kapitel 3.4). Beispiele derartiger Teile sind z.B. die oben aufgeführten, leeren Subsysteme, innerhalb derer die zukünftigen Frameworkinstanziierungen plaziert werden, und einige Klassen und Subsysteme, deren Zweck primär im beispielhaften

Aufzeigen der JWAM-Verwendung liegen. Die Systemgrößen werden im folgenden für die derart reduzierte Version und die vollständige Version (Werte in Klammern) angegeben:

- 72 (93) Subsysteme mit zusammen 787 (963) Dateien,
- 1159 (1456) Klassen mit zusammen 6746 (7953) Methoden und 1883 (2546) Attributen,
- 1054 (1297) Vererbungsrelationen zwischen den Klassen,
- 14738 (16674) Benutztbeziehungen zwischen Methoden und 5223 (5722) Benutztbeziehungen zwischen Methoden und Attributen.

Ein wesentliches Ergebnis der Auswertung und der Diskussionen war die Notwendigkeit einer detaillierteren Analyse der JWAM-Vererbungsstruktur. Indizien dafür waren vor allen Dingen die auf der einen Seite oberflächliche Betrachtung von Vererbung innerhalb der Menge verwendeter Maße (vgl. Abschnitt 9.2.1) und die auf der anderen Seite festgestellte häufige Verwendung des Vererbungskonzepts. Eine wichtige Voraussetzung für diese vertiefte Analyse ist die Vorbereitung einer weiteren Sicht auf das System, innerhalb derer jede Klasse vollständig, d.h. in der durch die Flatten-Technik modifizierten Sicht betrachtet und vermessen wird (vgl. Kapitel 8.2). Für diesen Zweck wurden alle Klassen, die Teil des Meßfokus waren und gleichzeitig Unterklassen besitzen, in den Vererbungskontext gesetzt, damit es bei der Definition des Qualitätsmodells möglich ist, auch geerbte Funktionalität, die von Klassen des Vererbungskontextes in alle Unterklassen eingeblendet wird, berücksichtigen zu können. Um diese Sicht entsprechend zu vermessen, wurden viele der in Abschnitt 9.2.1 aufgeführten Softwaremaße zweimal angewendet: Einmal für die normale Sicht auf das Systems, innerhalb derer eine Klasse unabhängig ihrer Vererbungen lediglich aus den in ihr definierten oder implementierten Methoden und Attributen besteht, und das andere Mal für die Sicht auf die durch die Flatten-Technik modifizierte Version. Die nach diesem Muster zusätzlich definierten Maße sind:

Abstraktions-niveau	Bezeichnung	Beschreibung
Subsystem	*P_EffIntCp_P_Flat*	Wie P_EffIntCp_P, allerdings werden alle Klassen aller Subsysteme inklusive der geerbten Funktionalität und deren ausgehende Interaktionen betrachtet. Polymorphe Strukturen werden um potentielle Interaktionskopplungen (vgl. Kapitel 8.2.2) erweitert.
	P_AffIntCp_P_Flat	Wie P_AffIntCp_P, allerdings werden alle Klassen aller Subsysteme inklusive der geerbten Funktionalität und deren einlaufende Interaktionen betrachtet. Polymorphe Strukturen werden ebenfalls um potentielle Interaktions-kopplungen erweitert.
Klasse	*C_PubM_Flat*	Wie C_PubM, allerdings werden ebenfalls geerbte, öffentliche Methoden berücksichtigt (vgl. Kapitel 8.2.1).
	C_PubA_Flat	Wie C_PubA, allerdings werden ebenfalls geerbte, öffentliche Attribute berücksichtigt (vgl. Kapitel 8.2.1).
	C_WMCoCh_Flat	Wie C_WMCoCh, allerdings werden ebenfalls geerbte Methoden berücksichtigt (vgl. Kapitel 8.2.1).
	C_EffIntCp_C_Flat	Wie C_EffIntCp_C, allerdings werden ebenfalls geerbte Funktionalität und deren ausgehende Interaktionen betrachtet. Polymorphe Strukturen werden um potentielle Interaktionskopplungen (vgl. Kapitel 8.2.2) erweitert.
	C_AffIntCp_C_Flat	Wie C_AffIntCp_C, allerdings werden ebenfalls geerbte Funktionalität und deren einlaufende Interaktionen berücksichtigt. Polymorphe Strukturen werden um potentielle Interaktionskopplungen erweitert.

Neben den damit für den zweiten Prozeßdurchlauf verwendeten 25 Maßen (18 aus Abschnitt 9.2.1 plus die obigen 7) wurden wiederum die notwendigen Daten für die gewichtete, methodenbasierte Interaktionskohäsion (s. Kapitel 8.3.2) und für die allgemeine Vererbungskohäsion (s. Kapitel 8.3.1) berechnet. Mit einem zur Beurteilung von JWAM 1.4 ähnlichen Ressourcenplan wurde die zweite Beurteilungsdurchführung begonnen:

Beurteilungsdurchführung
Der in diesem Teilprozeß entstandene Report umfaßt 45 Seiten mit 37 Eindrücken und 32 Restrukturierungsempfehlungen, die jeweils wie im Bericht zu JWAM 1.4 aufgebaut sind. Bereits während der Beurteilungsdurchführung und der damit verbundenen Reporterstellung hat sich folgendes, nicht weiter quantifiziertes Muster gebildet, das Aufschluß über den Zusammenhang zwischen den beiden Reporten gibt:

- *Repariert*: Viele Eindrücke und Empfehlungen, die auf qualitative Schwachpunkte deuteten und innerhalb der Beurteilung als richtig eingestuft wurden, erschienen nicht mehr im neuen Report, da die betroffenen Komponenten entweder vollständig gelöscht wurden – wie dies z.B. für einige Standardpakete der Fall war, die innerhalb der Version 1.4 sehr negativ auffielen und in der neuen Version 1.5. durch andere, z.T. selbst geschriebene Pakete ersetzt wurden – oder überarbeitet wurden (z.B. Beseitigung einiger zu hoch angesetzten Sichtbarkeiten von Methoden und Attributen).
- *Nachgerückt*: Aufgrund des Reparierens (s.o.) vieler besonders negativer Stellen konnten einige neue Qualitätsdefizite identifiziert werden, die zwar bereits aus den Meßdaten zu JWAM 1.4 hätten extrahiert werden können, dort allerdings durch die weitaus schlechteren, nun reparierten Stellen überdeckt wurden.
- *Neu*: Einige Anmerkungen sind gegenüber dem Report von JWAM 1.4 vollständig neu, weil entweder die entsprechenden Softwareteile neu hinzugekommen sind oder die Anmerkung erst durch die zusätzliche Sicht auf die durch die Flatten-Technik modifizierte Version des Systems identifizierbar wurde.
- *Übernommen*: Einige Anmerkungen, die auf qualitative Schwachpunkte im JWAM 1.4 hingewiesen haben und als richtig eingestuft wurden, konnten auch bei der erneuten Beurteilung identifiziert werden.

Dieses Muster wurde zusammen mit den quantitativen Eindrücken in der anschließenden Reflexion von den JWAM-Entwicklern bestätigt.
Auch dieser Report ist zusammen mit dem Auswertungsbogen zwei Wochen nach Eingang der Software an die JWAM-Entwickler geschickt worden.

Beurteilungsreflexion
Für die Validierung des verwendeten Prozesses mit den darin verwendeten Maßen war wiederum die von den JWAM-Entwicklern durchgeführte Bewertung der einzelnen Anmerkungen notwendig. Das Ergebnis ist in Abbildung 60 dargestellt[32]:

[32] Der qualitative Eindruck Nr. 30 wurde von der JWAM-Gruppe inhaltlich nicht verstanden, so daß er auch nicht bewertet werden konnte. Alle weiteren Analysen beziehen sich daher auf 32 Empfehlungen und lediglich 36 Eindrücke.

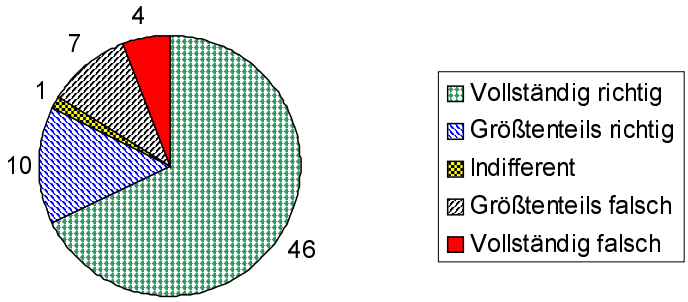


Abbildung 60: Bewertung der Anmerkungen zu JWAM 1.5

Auch diese Ergebnisse, die eine über 82-prozentige Zustimmung der Entwickler zur gemachten Qualitätsbewertung darstellen, belegen die hohe Validität und Effizienz des gewählten Ansatzes. Wie bereits bei der Auswertung von JWAM 1.4 kann dieser Eindruck durch eine Verfeinerung in Eindrücke und Restrukturierungsempfehlungen noch weiter untermauert werden, wie dies in Abbildung 61 dargestellt ist.

Abbildung 61: Ergebnisse der Auswertung für Eindrücke (links) und Empfehlungen (rechts)

Die über 97-prozentige Zustimmung der Entwickler zu den Eindrücken ist wie bereits bei der Reflexion des JWAM 1.4-Reports ein eindrucksvoller Beleg für die abermalige Validierung des gewählten Ansatzes. Der deutlich gesunkene Grad der Zustimmung bzgl. der gemachten Restrukturierungsempfehlungen hängt im wesentlichen von neu gemachten Empfehlungen ab, deren Identifikation erst durch die zusätzliche Sicht auf die durch die Flatten-Technik modifizierte Version des Systems möglich wurde. Da der zentrale Aspekt des Flatten die Berücksichtigung von Vererbung ist, die innerhalb des ersten Durchlaufs nur oberflächlich behandelt wurde und damit auch noch nicht Thema der Qualitätsmodellanpassung sein konnte, tritt hier abermals ein deutlicher Unterschied zwischen den Qualitätsvorstellungen der JWAM-Gruppe und der bzgl. der Vererbung standardmäßig angenommenen Qualitätsmerkmale, mit denen die Meßmaschinerie angepaßt wurde, zutage. So liegen die Beweggründe für alle 4 als vollständig falsch eingestuften Empfehlungen in der zusätzlich geschaffenen Sicht auf das durch die Flatten-Technik modifizierte System, und hier vor allen Dingen aus der daraus entstandenen, zusätzlichen Kopplung (vgl. Hinzufügen neuer Kopplungen durch Erben entsprechender Methoden in Kapitel 8.2.1 und potentielle Interaktionskopplungen in Kapitel 8.2.2).

Als Beispiel, dessen Charakter auf die anderen 3 Empfehlungen übertragbar ist, dient die als vollständig falsch beurteilte Restrukturierungsempfehlung Nr. 12:

Typ	Dominantes Maß	Anmerkung (dem Report entnommen)
Empfehlung	*P_EffIntCp_P_Flat*	*„Die Pakete* [...] *fallen durch eine übergroße Kopplung zu anderen Subsystemen auf. Grundsätzlich sollte Polymorhie über viele Paketgrenzen hinweg sehr vorsichtig angewendet werden. Wünschenswerter wäre, die Superklassen in das eigene Subsystem zu verschieben (z.B. innerhalb der* [...] *alle Superklassen von* [...] *und* [...]*, die sich momentan in einem anderen Subsystem befinden).“*

Die innerhalb des Qualitätsmodells gemachte Annahme, Vererbungsstrukturen, die für den Zweck der Schnittstellenimplementierung oder der Typhierarchieimplementierung (vgl. Kapitel 8.2) aufgebaut wurden, in ein gemeinsames Subsystem abzulegen, stand diametral zu einem erst nach Auswertung des zweiten Reports kommunizierten Entwurfsprinzip innerhalb der JWAM-Entwicklung für JWAM 1.5, Schnittstellen und deren Implementierung – wenn möglich – in verschiedene Subsysteme abzulegen. Die Anwendung dieses Prinzips führt einerseits zu schnittstellenlastigen Subsystemen, deren Inhalte primär für Client-Klassen wichtig sind, und implementierungslastigen Subsystemen, deren Inhalte primär für die Kodierarbeit, Versionierung und Testdurchführung der JWAM-Entwickler wichtig sind; andererseits führt dieser Ansatz aber zu einer starken, subsystemübergreifenden Kopplung, die schwierig nachzuvollziehen und damit schwer zu warten ist, da für das Verständnis eines Systemteils zwangsläufig mehrere Subsysteme betrachtet werden müssen. Die Einarbeitung dieser speziellen Entwurfsregel in Form eines weiteren angepaßten Qualitätsmodells in einen fiktiven dritten Prozeßdurchlauf würde die Akzeptanz des Reports noch weiter erhöhen.

Bezüglich der Frage nach dem Neuigkeitsgehalt der Anmerkungen im neuen Report mußte eine weitere Beantwortungsmöglichkeit hinzugefügt werden: „Bekannt aus letzter Vermessung“, d.h. Eindrücke und Empfehlungen, die bereits aufgrund des Reports zur Version 1.4 bekannt geworden sind, aber nach wie vor Gültigkeit besitzen. Die Ergebnisse sind in Abbildung 62 zusammengefaßt:

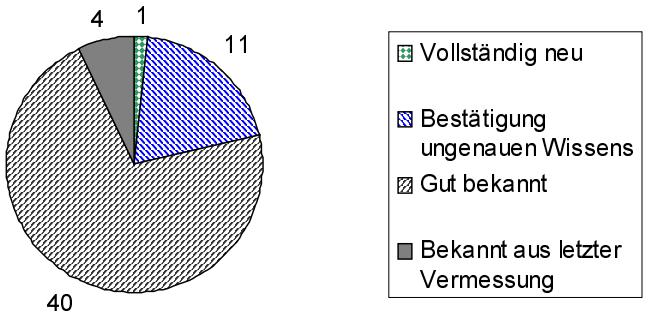


Abbildung 62: Bewertung des Neuigkeitsgehalts der als richtig beurteilten Anmerkungen des Reports

Der relativ geringe Grad, zu dem der Report Neuigkeiten vermittelt (über 71% der Anmerkungen sind gut bekannt), resultiert auch hier wieder aus der besonderen Situation des JWAM-Projekts (vgl. Abschnitt 9.2.1) und kann – für den eigentlichen Zweck der Prozeß- und Maßvalidierung – als Beleg für eine fachlich sehr fundierte Beurteilung der Anmerkungen herangezogen werden.

Die Ergebnisse beider Prozeßdurchläufe belegen eine erfolgreiche externe Validierung des gewählten Prozesses zur qualitativen Softwareproduktbeurteilung inklusive der verwendeten Techniken und Maße: Entsprechend der Definition der externen Validierung konnte damit

gezeigt werden, daß „[...] *die statistischen Operationen innerhalb des formalen Relationensystems und die dort stattfindende Interpretation die dem Messen zugrundeliegende Fragestellung beantworten hilft*“ (Definition „Externe Validierung“, Kapitel 4.3); dafür wurden folgende Parameter verwendet:

- *Zugrundeliegende Fragestellung*: Die dem Messen zugrundeliegende Fragestellung war die Frage nach der internen Qualität eines gegebenen Softwareprodukts. Für die Validierung selbst muß diese Frage allerdings fiktiv gestellt sein, d.h. es muß zusätzlich andere, mit wenig Aufwand behaftete Lösungen zur Beantwortung geben; anderenfalls ist die Überprüfung der meßwertbasierten Ergebnisse nicht möglich, da es kein Wissen über den tatsächlichen Ist-Zustand gibt. Nur die aus diesem Experiment gewonnene Erkenntnis, daß eine meßwertbasierte Analyse dem detaillierten und über softwaretechnische Belange hinausgehenden Systemwissen der Projektentwickler entspricht, ermöglicht es in nachfolgenden Anwendungen, auf eine derartige Referenz verzichten zu können, d.h. den Ansatz z.B. auf Kontexte zu übertragen, in denen ein derartiges Wissen nicht (mehr) existiert.
- *Statistische Operationen*: Die statistischen Operationen beschränken sich entsprechend des Vorgehens zur Interpretation der erstellten dreidimensionalen Informationsräume (vgl. Abschnitt 9.2) auf die Perzentilbildung (z.B. die 10 größten Werte) und die ordinale Einschätzung von Abständen zwischen Entitätspaaren, d.h. es wird lediglich eine Ordinalskala zugrunde gelegt.
- *Interpretation*: Die stattgefundene Interpretation beschränkt sich ebenfalls auf die Möglichkeiten innerhalb der Ordinalskala. Ein Nebeneffekt dieses Vorgehens ist, daß der gewählte Prozeß immer Schwachpunkte identifiziert, da die Perzentilbildung unabhängig absoluter Zahlen definiert ist. Dieser Effekt ist deutlich geworden in der zweiten Vermessung von JWAM, innerhalb derer einige Entitäten mit den größten Werten repariert worden sind. Das dadurch mögliche „Nachrücken“ anderer Entitäten belegt das Verwenden der Ordinalskala.

  Je mehr derartige Prozeßdurchläufe stattfinden und je mehr diesbezügliche Erfahrungen gesammelt werden, desto eher ist es möglich, projektübergreifende Vergleiche anzustellen. Innerhalb des empirischen Relationensystems führen diese zusätzlichen Erfahrungen zu vertieftem Wissen über große, objektorientierte Programme, d.h. über deren typische Werte bzgl. einiger Eigenschaften. Darauf aufbauend sind dann Interpretationen möglich, die, unter Einbeziehung der unterschiedlichen Einflußfaktoren (vgl. Kapitel 2.2), auf absoluten Zahlen basieren.

Eine weitere externe Validierung, innerhalb derer zusätzliche Kohäsionsformen angewendet wurden, die u.a. aber auch für den letzten Punkt, dem Ausbauen der Interpretationsbasis, durch Aufbau umfangreicher, viele unterschiedliche Projekte abdeckender Meßwertsammlungen dienlich sein kann, wird im folgenden Abschnitt vorgestellt.

9.2.3 Erster Prozeßdurchlauf für das Versicherungssystem VS

Ein weiteres Experiment für die Validierung des Meßansatzes ist für ein großes JAVA-System durchgeführt worden, das in einer großen deutschen Versicherung für die Abwicklung von Schadenserfassungen eingesetzt wird[33]. Die Funktionalität kann grob in die beiden Bereiche *Datenerfassung*, die als Aufgabe die korrekte Erfassung von für die spätere Schadensregulierung relevanten Parametern hat und von Versicherungsvertretern vor Ort mit eigenen Rechnern durchgeführt wird, und *Datenabgleich*, der als Aufgabe die Übertragung der eingegebenen Daten an einen Zentralrechner hat und damit die Schadensregulierung initiiert, aufgeteilt werden.

[33] Alle Daten, die entlang dieses Projekts entstanden sind, werden anonymisiert wiedergegeben; das System selbst wird im folgenden mit der fiktiven Bezeichnung *VS* bezeichnet.

Zur Entwicklung von VS: Eine Gesamtlösung für die Versicherungsvertreter vor Ort, innerhalb derer VS eine Teilkomponente darstellt, sollte durch die an die Versicherung angeschlossene EDV-Abteilung entwickelt werden. Mit der Erstellung des VS selbst wurde aufgrund personeller Engpässe allerdings eine außenstehende Firma beauftragt, d.h. VS wurde als *Outsourced Software* erstellt (vgl. Kapitel 3.3.1 und [Jone00]). Die vertragliche Grundlage des Projekts sah vor, daß für die nach Ablieferung anfallenden Wartungsaktivitäten die EDV-Abteilung selbst zuständig ist; für diesen Zweck begleiteten zwei Mitarbeiter der EDV-Abteilung die letzten Monate der VS-Entwicklung innerhalb des beauftragten Subunternehmens.
Mit der Auslieferung von VS ergab sich neben des funktionalen Abnahmetests für das Management die Frage nach der internen Qualität des Produkts, da diese maßgeblich die Folgekosten der Wartung beeinflussen. Um dieser Frage detailliert nachzugehen, wurde auf Wunsch des Managements der EDV-Abteilung eine meßbasierte Qualitätseinschätzung erstellt. Grundlage dafür war wiederum der in Abbildung 56 dargestellte Prozeß und dessen Ablaufparameter.

Beurteilungsvorbereitung
Der Quelltext zu VS wurde erst nach sorgfältiger Ausarbeitung eines Non-Disclosure-Agreements (vgl. Abschnitt 9) für die Vermessung zugänglich gemacht. Diese Daten waren für den ersten Prozeßdurchlauf zugleich die einzigen Daten.
Bereits während der mit Crocodile durchgeführten Produktmodellerstellung von VS, das vollständig in JAVA implementiert ist, sind folgende Systemgrößen identifiziert worden:

- 113 Subsysteme mit zusammen 1118 Dateien,
- 1182 Klassen mit zusammen 10556 Methoden und 4717 Attributen,
- 1049 Vererbungsrelationen,
- 24691 Benutztbeziehungen zwischen Methoden und 12798 Benutztbeziehungen zwischen Methoden und Attributen.

Aufgrund des Fehlens weiterer Informationen wurde für den ersten Durchlauf das gesamte Projekt vermessen und beurteilt. Aufgrund der positiven Erfahrungen mit dem Flatten-Konzept während der zweiten JWAM-Qualitätseinschätzung (vgl. Abschnitt 9.2.2) wurden allerdings von vornherein alle Klassen, die Unterklassen besitzen, in den Vererbungskontext gesetzt (vgl. Abschnitt 9.2.2).
Für den ersten Prozeßdurchlauf wurde wiederum ein Standard-Qualitätsmodell für große JAVA-Systeme verwendet, allerdings angereichert bzw. modifiziert durch die beiden Prozeßdurchläufe für das JWAM-System. Die innerhalb dieses Qualitätsmodells mit Beziehungen zu Qualitätskriterien versehenen Softwaremaße können wiederum in Größen- und Kopplungsmaße klassifiziert werden:

- Größenmaße: Die folgenden 13 Größenmaße wurden für die Qualitätsbeurteilung vom VS verwendet:

Abstraktions-niveau	Bezeichnung	Beschreibung
Sub-system	*P_Cl*	vgl. Abschnitt 9.2.1
Datei	*F_BLoC*	vgl. Abschnitt 9.2.1
	F_Del	vgl. Abschnitt 9.2.1
	F_CyclN	vgl. Abschnitt 9.2.1
	F_Comm	vgl. Abschnitt 9.2.1
	F_Import	Das Maß *F_Import* bestimmt für eine Datei die Anzahl der Import-Anweisungen. Es wird nicht zwischen allgemeinen Imports (`import <Paket>.*`) und spezifischen Imports (`import <Paket>.<Klasse>`) unterschieden.
Klasse	*C_PubM*	vgl. Abschnitt 9.2.1
	C_PubM_Flat	vgl. Abschnitt 9.2.2
	C_PubA	vgl. Abschnitt 9.2.1
	C_PubA_Flat	vgl. Abschnitt 9.2.2
	C_WMCoCh	vgl. Abschnitt 9.2.1
	C_WMCoCh_Flat	vgl. Abschnitt 9.2.2
Methode/ Attribut	*M_Ch*	Das Maß *M_Ch* bestimmt für jede Methode die Anzahl von Zeichen, die für ihre Implementierung benötigt werden. Innerhalb der durch „{“ eingeleiteten und durch „}“ beendeten Implementierung werden alle Zeichen, d.h. inkl. Leerzeichen, Zeilenumbrüche, Kommentare u.ä., betrachtet. Lediglich nur deklarierte Methoden besitzen einen M_Ch-Wert von 0.

- Kopplungsmaße: Die folgenden 15 Kopplungsmaße wurden für die Qualitätsbeurteilung vom VS verwendet:

Abstraktions-niveau	Bezeichnung	Beschreibung
Sub-system	*P_EffIntCp_P*	vgl. Abschnitt 9.2.1.
	P_EffIntCp_P_Flat	vgl. Abschnitt 9.2.2
	P_AffIntCp_P	vgl. Abschnitt 9.2.1
	P_AffIntCp_P_Flat	vgl. Abschnitt 9.2.2
Klasse	*C_Sup_C*	vgl. Abschnitt 9.2.1
	C_EffIntCp_C	vgl. Abschnitt 9.2.1
	C_EffIntCp_C_Flat	vgl. Abschnitt 9.2.2
	C_EffIntCp_M	Das Maß *C_EffIntCp_M* bestimmt für eine Klasse, zu wieviel Methoden anderer Klassen Interaktionen (s. P_EffIntCp_P) bestehen. Es gilt grundsätzlich für eine Klasse *C*: C_EffIntCp_M (*C*) ≥ C_EffIntCp_C (*C*).
	C_EffIntCp_M_Flat	Wie *C_EffIntCp_M*, allerdings werden ebenfalls geerbte Funktionalität der betrachteten Klasse, deren ausgehende Interaktionskopplungen und durch Methoden der betrachteten Klasse potentiell aufgerufene Methoden betrachtet.
	C_AffIntCp_C	vgl. Abschnitt 9.2.1
	C_AffIntCp_C_Flat	vgl. Abschnitt 9.2.2
	C_AffIntCp_M	Das Maß *C_AffIntCp_M* bestimmt für eine Klasse die Anzahl Methoden anderer Klassen, die zur betrachteten Klasse Interaktionen (s. P_EffIntCp_P) besitzen. Es gilt grundsätzlich für eine Klasse *C*: C_AffIntCp_M (*C*) ≥ C_AffIntCp_C (*C*).
	C_AffIntCp_M_Flat	Wie *C_AffIntCp_M*, allerdings werden ebenfalls geerbte Funktionalität der betrachteten Klasse, deren einlaufende Interaktionskopplungen und potentielle Kopplungen, die durch eine Oberklasse der betrachteten Klasse möglich sind, betrachtet.
Methode/ Attribut	*A_AffIntCp_OM*	vgl. Abschnitt 9.2.1
	A_AffIntCp_FM	vgl. Abschnitt 9.2.1

Zusätzlich zu diesen klassischen Maßen wurden für die anschließende Visualisierung neben den Daten für die bereits innerhalb des JWAM-Experiments erfolgreich validierte gewichtete, methodenbasierte und die allgemeine Vererbungskohäsion zusätzlich die Daten für eine gewichtete, attributbasierte Interaktionskohäsion bestimmt (vgl. Kapitel 8.3.2).
Mit einem dem JWAM-Experiment ähnlichen Ressourcenplan (vgl. Abschnitt 9.2) wurde anschließend die Beurteilungsdurchführung begonnen.

Beurteilungsdurchführung
Der in diesem Teilprozeß entstandene Report umfaßt 48 Seiten mit 26 Restrukturierungsempfehlungen und 34 Eindrücken. Die Anmerkungen sind innerhalb des Reports in Anlehnung an die JWAM-Reporte strukturiert, d.h. jede Anmerkung wird einem dominanten Maß zugesprochen.

Der Report ist zusammen mit einem wiederum an das JWAM-Experiment angelehnten Auswertungsbogen zwei Wochen nach Eingang der Software an die EDV-Abteilung der Versicherung geschickt worden.

Beurteilungsreflexion
Statt des ausgefüllten Fragebogens wurde von den beiden Entwicklern, die die Software die letzten beiden Monate innerhalb des Subunternehmens begleitet haben, eine 7-seitige, umgangssprachliche Auswertung der meßwertbasierten Analyse erstellt, die jede einzelne Anmerkung separat behandelt. Um über dieses sehr ausführliche Dokument einen groben Überblick zu erhalten, ist das Abbilden der erhaltenen, offenen Antworten auf die geschlossenen Antworten (vgl. Abschnitt 9.2) nachträglich geschehen und kann wie in Abbildung 63 dargestellt werden:

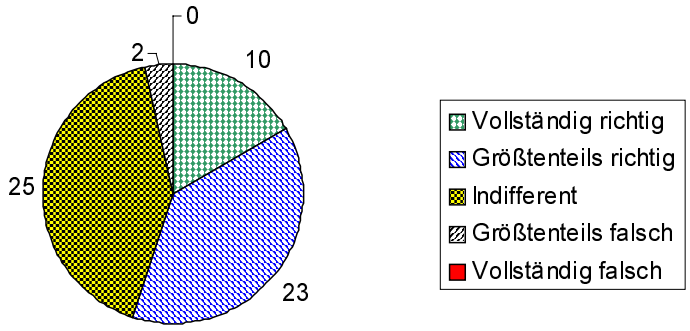


Abbildung 63: Bewertung der Anmerkungen zu VS

Der größte Anteil (~42%) entfiel dabei auf die Antwort „indifferent“, in die folgende Arten der umgangssprachlichen Auswertungen klassifiziert wurden:

- *Verständnisfragen*: Viele Anmerkungen wurden nicht verstanden und daher innerhalb der Beurteilung lediglich mit Fragen für ein vertieftes Verständnis kommentiert. Typische Ausprägungen waren „*Wie kommen diese Ergebnisse zustande?*“, „*Ist das gut oder schlecht?*“ und „*Es ist schwer einzuschätzen, welche Aussagekraft die Werte der geflatteten Version wirklich haben*“ (jeweils aus der Auswertung zitiert).
- *Vorgehenskritik*: Einige Anmerkungen wurden nicht bewertet, sondern es wurden der gesamte Ansatz einer meßbasierten Analyse und die dort verwendeten Maße kritisiert. Typische Ausprägungen für einzelne Maße waren Grundsatzdiskussionen über die Aussagekraft der F_BLoC-Werte und die Möglichkeiten, diese Werte künstlich – z.B. unter Verwendung des `?`-Operators anstelle einer ausgeschriebenen `if-then-else` Anweisung – zu verbessern, sowie Infragestellungen der Praktikabilität einiger auf Basis des hierfür dominanten Maßes P_EffIntCp_P begründeten Restrukturierungsempfehlungen hinsichtlich der vor Ort verwendeten Werkzeuge (z.B. für Persistenz). Typische, das gesamte Vorgehen kritisierende Antworten bezogen sich vor allen Dingen auf Anmerkungen, die generierte Softwareteile betrafen, da diese für den konkreten Kontext der Qualitätsbewertung als irrelevant eingeschätzt wurden.
- *Rechtfertigung*: Einige Anmerkungen wurden gar nicht erst bewertet, sondern ihre Ursachen unmittelbar gerechtfertigt. Typische Ausprägungen waren „*Dieses Verhalten ist durchaus legitim, da sie genau für diesen Zweck geschaffen wurden*“, „*Fast alle genannten Methoden sind dispatch-Methoden und stammen aus den generierten Skeleton-Methoden (RMI)*“ und „ *Die Konstanten müssen im Interface deklariert sein, damit Zugriffe darauf unabhängig vom tatsächlich zur Verfügung stehenden Objekt* [...] *ermöglicht werden*“ (jeweils aus der Auswertung zitiert).

Alle drei Typen von nachträglich als indifferent eingestuften Bewertungen resultieren aus der besonderen Projektsituation der durchgeführten Qualitätsbewertung und belegen in praktischer

Form viele in Kapitel 3.6. aufgeführten Probleme beim Produktmaßeinsatz. Für einige Dimensionen der allgemeinen Probleme beim Einsatz neuer Techniken (vgl. Kapitel 3.6.1) bedeutet dies konkret :

- *Vorgehen*: Die Qualitätsbewertung wurde vom Management initiiert. Das Ergebnis (in diesem Fall der Report) wurde anschließend den Entwicklern vorgelegt, denen zu diesem Zeitpunkt weder realistische Ziele noch erhoffter Nutzen bekannt waren.
- *Management*: Das Management hat weder den mit der Vorlage der fertigen Qualitätsbewertung verbundenen Mehraufwand geplant, noch für eine nötige Unternehmenskultur für den offenen Umgang von durch Dritte gefundene, meßbasierte Ergebnisse gesorgt. Jede Anmerkung bzgl. einer einer konkreten Person zuzuordnenden Entität wurde daher als persönlicher Angriff gewertet.
- *Promotoren*: Außer dem das Projekt startenden Management gab es keinerlei Promotoren für die Qualitätsbewertung. Zusammen mit der Unkenntnis der Entwickler bzgl. der Beweggründe der Qualitätsbewertung ergab sich daraus eine unkontrollierte und zwischen den Entwicklern unterschiedliche Herangehensweise an den Report.
- *Betroffene*: Die Art der Projektinitiierung und Durchführung hat viele Entwickler eine grundsätzliche Abwehrhaltung gegenüber dem Report einnehmen lassen. Es war weder klar, daß nicht die Entwickler selbst vermessen wurden, noch wurde Rücksicht auf eigene, bereits etablierte Qualitätsmaßnahmen genommen.
- *Ausbildung und Unterstützung*: Speziell die vielen Verständnisfragen offerierten einen enormen Nachholbedarf an Ausbildung bzgl. objektorientierter Softwareentwicklung. Für viele als ausgewiesene COBOL-Kenner bezeichnete Entwickler war VS das erste objektorientierte System. Selbst die beiden Entwickler, die die Software die letzten beiden Monate innerhalb des Subunternehmens begleiteten, haben objektorientierte Softwareentwicklung in hauseigenen Intensivkursen und anhand der Programmiersprache *Smalltalk* erhalten. Speziell die Flatten-Technik und die damit verbundene neue Sicht auf das System stellte für viele ein großes Verständnisproblem dar.

Für die konkreten Probleme des Produktmaßeinsatzes (vgl. Kapitel 3.6.2) wiegt besonders schwer die fehlende Anpassung des Qualitätsmodells an konkrete Vorgaben. Dies umfaßt vor allen Dingen folgende Punkte:

- *Systemauswahl*: Da das Management keine detaillierten, technischen Systemkenntnisse besitzt, wurde das komplette System, so wie es später auch eingesetzt werden sollte, übermittelt und vermessen. Daß einige Teile des VS wiederverwendet oder generiert sind, ist erst durch die Auswertung zutage getreten.
- *Umgebungsparameter*: Die Verwendung von speziellen Werkzeugen im Kontext des VS, wie z.B. ein eingesetztes Persistenz-Werkzeug, das lediglich öffentlich zugängliche Attribute persistent ablegen kann, wurde verschwiegen, so daß einige Aspekte des Standard-Qualitätsmodells von falschen Annahmen ausgingen.
- *Entwurfsprinzipien*: Die während der VS-Entwicklung eingehaltenen Entwurfsprinzipien wurden ebenfalls verschwiegen (z.B. die Forderung, für jeden auftretenden Exception-Typ eine eigene Klasse zu bilden), so daß auch hier das Standard-Qualitätsmodell wieder von falschen Annahmen ausging.

Alle hier aufgeführten Probleme wurden in einem direkten, eintägigen Gespräch vor Ort geklärt. Ein wesentliches Ergebnis, für das der Report selbst nur ein Mittel zum Zweck darstellte, war eine über die verschiedenen Unternehmensebenen hinweg stattfindende Diskussion über Softwarequalität und daraus resultierende, organisatorische Maßnahmen für deren Verbesserung.
Erst nach einem bzgl. dieses Themas erreichten groben Konsenses wurden die Ziele und damit verbunden wichtige Parameter für einen zweiten Meßdurchlauf erörtert. Die Ergebnisse waren

eine detaillierte Angabe von Systemteilen, die explizit für die Qualitätsbewertung ausgeblendet werden sollten, eine vertiefte Analyse der Vererbung inklusive eines erweiterten Flatten-Konzepts (s.u.) und einige Maßmodifikationen (s.u.).
Ein zweiter Prozeßdurchlauf, der wiederum in Anlehnung an den in Abbildung 56 dargestellten Prozeß durchgeführt wurde, hat versucht, diese Ergebnisse bestmöglich einzuarbeiten.
Die Aussagekraft dieses ersten Prozeßdurchlaufs als Vorvalidierung ist aufgrund der konkreten Projektprobleme (s.o.) reduziert: Werden die 25 als indifferent eingestuften Antworten nicht berücksichtigt, so werden immerhin über 94% der in diese Bewertung einfließenden 35 Anmerkungen als korrekt eingestuft; werden die 25 Antworten als keine Zustimmung interpretiert, so sind dies nur noch 55%. Die mit den Entwicklern geführten Diskussionen sprechen allerdings eher für die erste Variante, da für viele als indifferent klassifizierten Anmerkungen die Frage, ob damit reale Fragestellungen gelöst werden können (vgl. externe Validierung, Kapitel 4.3.2), aufgrund des speziellen Umfelds nicht geklärt werden konnte.

9.2.4 Zweiter Prozeßdurchlauf für das Versicherungssystem VS

Direkt nach der Reflexion des ersten Reports wurde ein neuer Prozeßdurchlauf gestartet. Im Gegensatz zum JWAM-Experiment wurde hierbei keine neue Version des Softwaresystems vermessen, sondern die bereits verfügbare Version mit geänderten Parametern neu beurteilt. Der Forderung der externen Validierung, nach der Vorvalidierung ein anderes Projekt zu vermessen, kann daher nur dadurch entsprochen werden, daß der neue Meßfokus große Teile des ursprünglichen Systems ausgeblendet hat (z.B. über 46% der Klassen), d.h. ein neues Projekt entstanden ist, das zwar vollständig Teil des ursprünglichen Systems ist, sich aber dennoch nach außen unterschiedlich präsentiert.
Die Beschreibung der zweiten Qualitätsbeurteilung vom VS geschieht wiederum wie in Abschnitt 9.2.3 entlang des aufgeführten Prozesses:

Beurteilungsvorbereitung

Die Hauptänderung gegenüber des ersten Prozeßdurchlaufs war eine sehr detaillierte Liste von Systemteilen, die innerhalb der zweiten Qualitätsbewertung nicht berücksichtigt werden sollten. Die deselektierten Systemteile, die auf der Ebene der Klassen angegeben wurden, können jeweils einer der folgenden sechs Funktionalitäten zugeordnet werden:

- *RMI*: Alle für die vom VS verwendete RMI-Kommunikation (*Remote Method Invocation*) notwendigen, generierten Klassen: Hierbei handelt es sich um sogenannte *Stub*–Klassen, die für das *Marshaling*, d.h. auf der Client-Seite für „[...] *das Verpacken der Operationsparameter für den Transport über Prozessgrenzen hinweg*“ ([GrTh00], S. 73) verantwortlich sind, und *Skeleton*-Klassen, die für das *Demarshaling*, d.h. auf der Server-Seite „[...] *für das Auspacken der Operationsparameter für den Empfänger*“ ([GrTh00], S. 73) verantwortlich sind.
- *EJB*: Alle für eine geplante Umstellung des VS auf *EJB*-Technik (*Enterprise Java Beans*) existierenden Klassen: Hierbei handelt es sich um sogenannte *Home*-Interfaces, die für jede EJB-Komponente angeboten werden müssen, um einem Client das Erzeugen oder Finden bereits existierender Instanzen der gewünschten Komponente zu ermöglichen (vgl. [GrTh00], S. 211ff), und *HomeImpl*-Implementierungen, „[...] *die vom jeweiligen EJB-Applikations-Server unter Einsatz von Introspektion automatisch generiert werden*“ ([GrTh00], S. 213).
- *BeanInfo*: Alle für das Anbieten von *Metainformationen* für die visuelle Programmierung notwendigen Klassen innerhalb der *JavaBean*-Entwicklung, die über die Anwendung der durch Namenskonventionen ermöglichten *JAVA-Reflection* hinausgehende Metainformationen für eine *Introspektion* anbieten: Hierbei handelt es sich um sogenannte *BeanInfo*-Klassen, die als Implementierung des *BeanInfo-Interfaces* Metainformationen u.a. über Ereignisse, Eigenschaften und Methoden einer JavaBean-Komponente anbieten ([GrTh00], S. 154ff).

- *Exception*: Alle lediglich für das Abfangen von Ausnahmen erstellten Klassen: Aufgrund einheitlich verwendeter Programmierrichtlinien während der VS-Erstellung ließen sich diese Klassen mittels einfacher regulärer Ausdrücke identifizieren.
- *COBOL*: Alle für das Kapseln von COBOL-Teilanwendungen notwendigen Klassen: Hierbei handelt es sich um von einem Werkzeug automatisch generierte Klassen, die die in COBOL geschriebenen *Copystrecken*, d.h. von Entwicklern entwickelte COBOL-Teile, die zur Compile-Zeit via des COPY-Mechanismus in das Programm eingefügt werden, in JAVA-Klassen verpacken (*wrappen*). Die Bezeichnungen dieser Klassen sind durch das Werkzeug festgelegt und konnten daher einfach mittels regulärer Ausdrücke gefunden werden.
- *VisualAge*: Alle visuell programmierten Komponenten, die das verwendete Werkzeug VisualAge größtenteils automatisch erstellt. Auch diese Klassen waren einfach mittels regulärer Ausdrücke auffindbar.

Das erneute, diese Ausblendungen berücksichtigende Erstellen des Produktmodells ergab folgende Größenwerte:

- 113 Subsysteme mit zusammen 593 Dateien,
- 646 Klassen mit zusammen 6105 Methoden und 2005 Attributen,
- 530 Vererbungsrelationen,
- 15077 Benutztbeziehungen zwischen Methoden und 5677 Benutztbeziehungen zwischen Methoden und Attributen.

Eine weitere wesentliche Änderung war eine vertiefte Analyse der Vererbungsstruktur mittels einer Verfeinerung der bzgl. des Flatten sensitiven Kopplungsmaße, die im folgenden anhand des Beispielsystems in Abbildung 64 erläutert wird: Jedes Maß, das auf der Interaktionskopplung aufbaut, wird 4-fach angewendet, wobei jede Anwendung eine andere Sicht des Systems betrachtet:

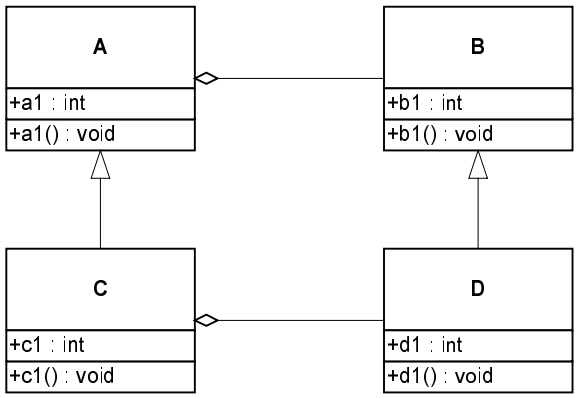


Abbildung 64: Beispiel für verfeinerte Sichten auf Vererbungsstrukturen

1. *Normale Sicht*: Diese entspricht der üblichen Struktur innerhalb von Klassendiagrammen: Für das Beispiel bedeutet dies, daß Klasse $\mathcal{A}$ lediglich Klasse $\mathcal{B}$ besitzt und Klasse $\mathcal{C}$ lediglich Klasse $\mathcal{D}$ besitzt.
2. *Potentielle Kopplung berücksichtigende Sicht*: In dieser Sicht werden ausschließlich potentielle Kopplungen berücksichtigt: Für das Beispiel bedeutet dies, daß die Klasse $\mathcal{A}$ die Klasse $\mathcal{D}$ besitzt, da aufgrund der Vererbungsbeziehung zwischen Klasse $\mathcal{B}$ und Klasse $\mathcal{D}$ jedes Objekt der Klasse $\mathcal{D}$ gleichzeitig ein Element der Klasse $\mathcal{B}$ ist. Klasse $\mathcal{C}$ besitzt in dieser Sicht keine Assoziation.
3. *Geerbte Funktionalität berücksichtigende Sicht*: In dieser Sicht wird ausschließlich geerbte Funktionalität betrachtet. Für das Beispiel bedeutet dies, daß die Klasse $\mathcal{A}$ keinerlei Interaktionen nach außen hat, da $\mathcal{A}$ keine Superklasse besitzt. Klasse $\mathcal{C}$ dagegen erbt das Aggregat vom Typ der Klasse $\mathcal{B}$, besitzt damit folglich die Möglichkeit, mit dieser Klasse zu interagieren.

4. *Flatten vollständig berücksichtigende Sicht*: Diese Sicht entspricht der Anwendung aller in Kapitel 8.2 aufgeführten Flatten-Aktivitäten auf das Ausgangssystem und entspricht den in Abschnitt 9.2.2 und 9.2.3 angewendeten Flat-Maßen. Demnach haben beide Klassen $\mathcal{A}$ und $\mathcal{C}$ Kopplungen zu beiden Klassen $\mathcal{B}$ und $\mathcal{D}$.

Ein Großteil der Veränderungen des für den zweiten Prozeßdurchlauf verwendeten Qualitätsmodells ergab sich aus diesen 4 Sichten auf das System. Darüber hinaus sind aber auch neue Anforderungen aus der Diskussion der ersten Qualitätsbewertung eingeflossen.
Insgesamt wurden folgende Kopplungsmaße für die zweite Qualitätsbewertung zusätzlich zu den in Abschnitt 9.2.3 aufgeführten Maßen verwendet:

Abstrak-tions-niveau	Bezeichnung	Beschreibung
Subsystem	*P_EffInhCp_P*	Das Maß *P_EffInhCp_P* bestimmt für ein Subsystem die Anzahl anderer Subsysteme, zu denen (über die jeweiligen Klassen der Subsysteme) Vererbungskopplungen bestehen[34]. Als Vererbungskopplung werden alle in Kapitel 8.2 aufgeführten Arten der Vererbung betrachtet.
	P_AffInhCp_P	Das Maß *P_AffInhCp_P* bestimmt für ein Subsystem die Anzahl anderer Subsysteme, die zum betrachteten Subsystem (über die jeweiligen Klassen der Subsysteme) Vererbungs-kopplungen (vgl. P_EffInhCp_P) besitzen.
	P_EffIntCp_P_Pot	Wie P_EffIntCp_P, allerdings wird die potentielle Kopplung berücksichtigende Sicht verwendet.
	P_EffIntCp_P_Inh	Wie P_EffIntCp_P, allerdings wird die geerbte Funktionalität berücksichtigende Sicht verwendet.
	P_AffIntCp_P_Pot *P_AffIntCp_Inh*	Wie P_AffIntCp_P, allerdings wird jeweils in Anlehnung an die Namensextension eine entsprechende Sicht verwendet.
	P_EffIntCp2_P	Wie P_EffIntCp_P, allerdings wird als Interaktion lediglich die direkte Attributsbenutzung betrachtet.
	P_EffIntCp2_P_Pot *P_EffIntCp2_P_Inh* *P_EffIntCp2_P_Flat*	Wie P_EffIntCp2_P, allerdings wird jeweils in Anlehnung an die Namensextension eine entsprechende Sicht verwendet.
	P_AffIntCp2_P	Wie P_AffIntCp_P, allerdings wird als Interaktion lediglich die direkte Attributsbenutzung betrachtet.
	P_AffIntCp2_P_Pot *P_AffIntCp2_P_Inh* *P_AffIntCp2_P_Flat*	Wie P_AffIntCp2_P, allerdings wird jeweils in Anlehnung an die Namensextension eine entsprechende Sicht verwendet.

[34] Um den Begriff der Vererbungskopplung in Anlehnung an Kapitel 7.2.3 hier weiter verwenden zu können, wird diese im folgenden als gerichtet betrachtet, d.h. es gibt einen aktiven Teil, von dem Vererbungskopplung ausgeht, d.h. der etwas erbt, und einen passiven Teil, von dem etwas geerbt wird. Das Maß P_EffInhCp_P bestimmt also die Anzahl der Subsysteme, von denen (über die jeweiligen enthaltenen Klassen) etwas geerbt wird, d.h. zu denen Vererbungskopplungen bestehen.

Klasse	*C_Sub_C*	Das Maß *C_Sub_C* bestimmt für eine Klasse die Anzahl direkter Unterklassen. Alle Formen der Vererbung werden gleichwertig betrachtet (vgl. Kapitel 8.2).
	C_EffIntCp_C_Pot *C_EffIntCp_C_Inh*	Wie C_EffIntCp_C, allerdings wird jeweils in Anlehnung an die Namensextension eine entsprechende Sicht verwendet.
	C_AffIntCp_C_Pot *C_AffIntCp_C_Inh*	Wie C_AffIntCp_C, allerdings wird jeweils in Anlehnung an die Namensextension eine entsprechende Sicht verwendet.
	C_EffIntCp2_C	Wie C_EffIntCp_C, allerdings wird als Interaktion lediglich die direkte Attributsbenutzung betrachtet.
	C_EffIntCp2_C_Pot *C_EffIntCp2_C_Inh* *C_EffIntCp2_C_Flat*	Wie C_EffIntCp2_C, allerdings wird jeweils in Anlehnung an die Namensextension eine entsprechende Sicht verwendet.
	C_AffIntCp2_C	Wie C_AffIntCp_C, allerdings wird als Interaktion lediglich die direkte Attributsbenutzung betrachtet.
	C_AffIntCp2_C_Pot *C_AffIntCp2_C_Inh* *C_AffIntCp2_C_Flat*	Wie C_AffIntCp2_C, allerdings wird jeweils in Anlehnung an die Namensextension eine entsprechende Sicht verwendet.

Darüber hinaus wurde das Größenmaß *F_Import* verfeinert in ein Maß *F_ImportSpec*, das alle vollständig spezifizierten Import-Anweisungen der Art `import <Paket>.<Klasse>` zählt, und in ein Maß *F_ImportCom*, das alle allgemeinen Import-Anweisungen der Art `import <Paket>.*` zählt.

Folgende Maße des ersten Prozeßdurchlaufs wurden aufgrund dessen Beurteilungsreflexion nicht wieder für das VS angewendet:

- *C_EffIntCp_M* und *C_EffIntCp_M_Flat*, da für das VS kein deutlicher Informationsgewinn gegenüber den weniger fein granularen C_EffIntCp_C-Maßen erwartet wurde,
- *C_AffIntCp_M* und *C_AffIntCp_M_Flat*, da auch hier das entsprechende, nur Klassen berücksichtigende Maß C_AffIntCp genügte, und
- alle Methoden- und Attributsmaße, da diese Betrachtung für das VS aufgrund der bereits auf höheren Abstraktionsebenen erwarteten signifikanten Ergebnisse als zu feingranular angesehen wurde.

Für die Kohäsionsbetrachtung wurden wiederum die bereits im ersten Prozeßdurchlauf angewendeten Kohäsionsarten errechnet und visualisiert.

Beurteilungsdurchführung
Der in diesem Teilprozeß entstandene Report umfaßt 54 Seiten mit 34 Eindrücken und 39 Restrukturierungsempfehlungen. Die Anmerkungen sind innerhalb des Reports in Anlehnung an den JWAM-Report strukturiert.
Aufgrund der festgestellten Verständnisprobleme bei vielen Maßen und deren Interpretationen wurde innerhalb des Reports ein eigenes Kapitel der vertieften Erläuterung der verwendeten Konzepte gewidmet; speziell das Konzept des Flatten und die vier unterschiedlichen Sichten wurden detailliert erläutert.
Der Report ist zusammen mit einem wiederum an das JWAM-Experiment angelehnten Auswertungsbogen zwei Wochen nach Beginn der zweiten Qualitätsbewertung an die EDV-Abteilung der Versicherung geschickt worden.

Beurteilungsreflexion
Die Ergebnisse des zurückgesandten, ausgefüllten Fragebogens, der wiederum von den beiden Entwicklern, die die Software die letzten beiden Monate innerhalb des Subunternehmens begleitet haben, beantwortet wurde, sind in Abbildung 65 dargestellt:

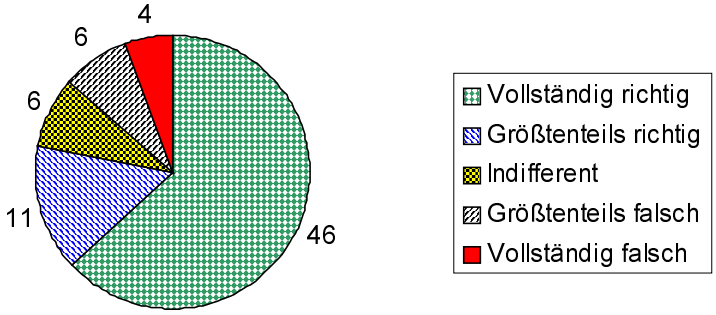


Abbildung 65: Bewertung der Anmerkungen zum Report der zweiten Qualitätsbewertung

Im Gegensatz zum ersten Report gab es keine Verständnisschwierigkeiten mehr: Von den 6 als indifferent bewerteten Anmerkungen sind 5 Restrukturierungsempfehlungen, deren Qualitätseindrücke zwar explizit als korrekt bewertet wurden, deren darauf aufbauende Restrukturierungsmaßnahmen aber nur mit erheblichem Aufwand – speziell hinsichtlich der Praktikabilität – bewertbar waren. Ein typisches solches Beispiel ist die als indifferent bewertete Restrukturierungsempfehlung Nr. 30:

Typ	Dominantes Maß	Anmerkung (dem Report entnommen)
Empfehlung	P_AffIntCp2_P (inkl. der drei zusätzlichen Sichten)	*„Die hohen Werte der Subsysteme* [...] *resultieren größtenteils aus dem Erben entsprechender Attribute (vgl. die geerbte Funktionalität berücksichtigende Sicht), die dann von vielen anderen Subsystemen verwendet werden. Auch hier fällt die strukturelle Gemeinsamkeit vieler verschiedener Subsysteme* [...] *sowie die exponierte Stellung einiger „besonderer" Subsysteme* [...] *auf. Diese „Besonderheit" wird durch das Gleichstellen mit den anderen Subsystemen* [...] *(alle sind direkte Untersubsysteme von dem besonderen Subsystem* [...]*) zunichte gemacht. Sinnvoller wäre das Gruppieren der ähnlichen Subsysteme zu den jeweils besonderen (z.B. .durch zusätzliche Subsysteme) wie z.B.* [...]."

Die Bewertung dieser, besonders in der Visualisierung deutlich zu erkennenden Situation wurde als korrekt bewertet, d.h. die aufgrund der Subsystemstruktur suggerierte Ähnlichkeit der Subsysteme wurde als ungünstig und dem Verständnis nicht hilfreich bewertet. Da diese Subsysteme allerdings zentrale Funktionalität bereitstellen, kann die vorgeschlagene Restrukturierungsempfehlung, die Subsystemstruktur neu zu bilden, nur nach aufwendiger, eingehender Untersuchung vor Ort beantwortet werden. Speziell der notwendige Umfang der Anpassung des für diesen Subsystemzweig angewendeten Persistenz-Werkzeugs war unklar.

Das positive Ergebnis dieser Auswertung, nach der über 78% der Anmerkungen als mindestens größtenteils richtig bewertet wurden, kann wieder durch die separate Betrachtung der Bewertungen zu Empfehlungen und Eindrücken (vgl. Abbildung 66) weiter untermauert werden:

Abbildung 66: Ergebnisse der zweiten VS-Auswertung für Eindrücke (links) und Empfehlungen (rechts)

Die über 91-prozentige Zustimmung der Entwickler zu den Eindrücken ist ein eindrucksvoller Beleg für die Validierung des verwendeten Ansatzes. Gleichzeitig wird deutlich, wie wichtig die im Vorfeld einer Qualitätsbewertung notwendige Kommunikation zwischen Entwicklungs- und Bewertungsteam ist. Die im Verhältnis dazu geringere 66-prozentige Zustimmung der Entwickler zu den Empfehlungen belegt nämlich, daß für diese Aktivität durchaus noch mehr Ressourcen hätten eingeplant werden müssen. So sind die meisten der als größtenteils falsch und vollständig falsch bewerteten Empfehlungen auf immer noch lückenhafte Kenntnisse der konkreten Umgebung des VS zurückzuführen. Speziell die Problematik, zwischen einem *fachlichen Entwurf*, d.h. einem Entwurf, der durch die Domäne der VS-Anwendung geprägt ist, und einem *technischen Entwurf*, der durch Umsetzung softwaretechnischer Anforderungen auf den fachlichen Entwurf entsteht, unterscheiden zu müssen, wurde immer noch deutlich unterschiedlich bewertet.
So wird innerhalb des VS z.B. die in Abbildung 67 dargestellte Vererbungsstruktur sehr intensiv eingesetzt, d.h. es existieren fast 30 Interfaces (in Abbildung 67 angedeutet durch das Präfix `Int_`) mit jeweils genau einer Implementierung. Das Motiv dieser Struktur, die fachliche Funktionalität (Klassen *B*, *C*, etc. und deren jeweilige Interfaces) auf diese Art und Weise mit technischer Funktionalität (gegeben durch die Klasse *A* und deren Interface) miteinander zu verschmelzen, kann durchaus unterschiedlich bewertet werden. So besitzen z.B. die Funktionalität tragenden Klassen *B*, *C*, etc. aufgrund der technischen Anforderungen alle eine gemeinsame Oberklasse; fachlich decken diese Klassen allerdings sehr unterschiedliche Aspekte ab. Diese Sensibilisierung für den teilweise auftretenden Bruch beim Übergang des fachlichen Entwurfs in einen technischen steht innerhalb

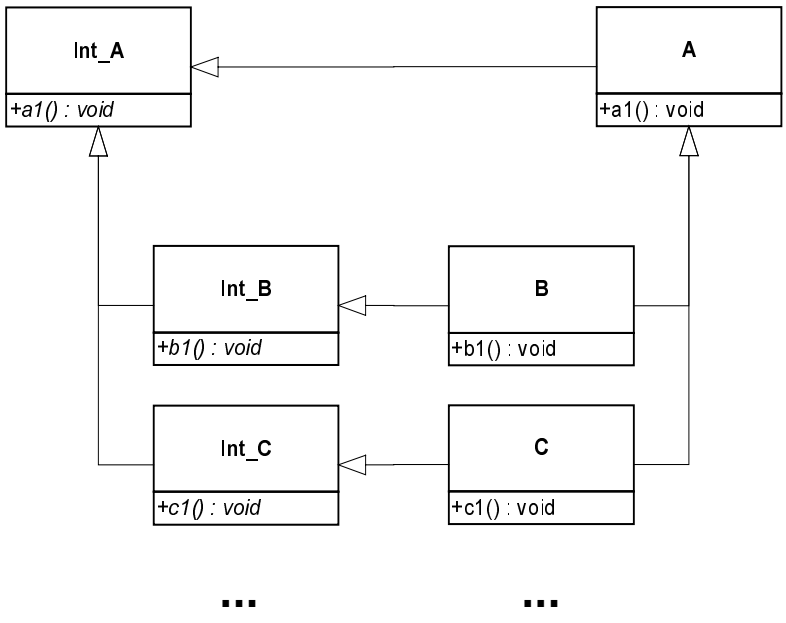


Abbildung 67: Beispiel einer Vermischung fachlicher und technischer Anforderungen

der VS-EDV-Abteilung erst am Anfang, so daß diesbezügliche Restrukturierungsempfehlungen (z.B. die Anbindung der technischen Funktionalität über eine Assoziation, d.h. eine Klasse B benutzt eine Klasse A, und/oder die Abtrennung der Interfaces der fachlichen Funktionalität vom Interface der technischen Funktionalität) abgelehnt wurden. Die durch diese Qualitätsbewertung innerhalb der Versicherung entstandene Diskussion trägt sicherlich positiv zu dieser Entwicklung bei.

Der Frage nach dem Neuigkeitsgehalt der Anmerkungen im neuen Report wurde mittels eines, in Anlehnung an den zweiten Prozeßdurchlauf vom JWAM-Experiment gestalteten Fragebogens nachgegangen. Die Ergebnisse sind in Abbildung 68 dargestellt.

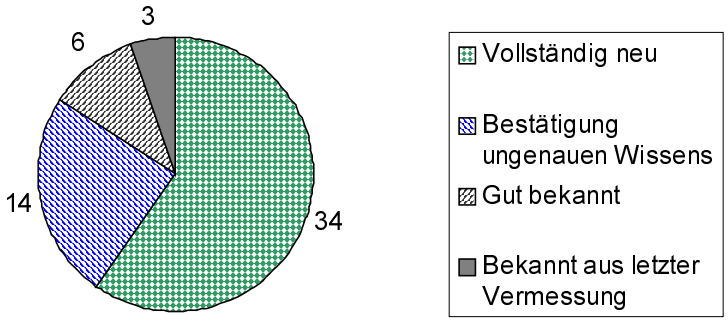


Abbildung 68: Bewertung des Neuigkeitsgehalts der als richtig beurteilten Anmerkungen des Reports

Diese Zahlen spiegeln einen deutlich anderen Wissensstand innerhalb der EDV-Abteilung als innerhalb des JWAM-Teams wider: Nur 24% der gemachten Anmerkungen wurden als gut bekannt bewertet. Damit besitzt dieser Report für die Versicherung einen deutlich höheren Wert, da die darin stattfindende Bewertung größtenteils neue oder nur ungenau bekannte Aspekte anspricht.
Wesentliche Gründe für diesen hohen Neuigkeitsgehalt sind:

- die beurteilenden Entwickler waren nicht von Anfang an Teil der VS-Entwicklung, woraus folgt, daß sie auch nur partiell detaillierte Kenntnisse über das VS besitzen,
- innerhalb der Versicherung existiert ein (noch) nur sehr rudimentäres Qualitätsverständnis von Softwareprodukten,
- eine noch im Aufbau befindliche Kompetenz innerhalb der EDV-Abteilung bzgl. objektorientierter Entwicklung und
- ein noch nicht auf die Erstellung qualitativ hochwertiger Software ausgerichtetes organisatorisches Umfeld (z.B. entsprechende Prozesse, Konventionen o.ä.).

Die für diese Arbeit primär relevante externe Validierung läßt sich dennoch aus der Korrektheit der Anmerkungen des zweiten Reports ableiten. Im Gegensatz zum JWAM-Experiment, bei dem die gefundenen Schwachstellen größtenteils bekannt und das Experiment damit primär der Validierung diente, konnte hier der Ansatz einen wirklich praxisrelevanten Wert aufzeigen. Die gemachte Validierung in Form der jeweils durchgeführten Reflexion hat diesen Wert nur noch einmal unterstrichen.

9.3 Externe Validierung innerhalb des Prozesses Überarbeitung eines Programms

Die externe Validierung des vorgestellten Meßansatzes entlang des Prozesses „Überarbeitung eines Programms“ soll entlang zweier Anwendungsszenarien erfolgen:

1. Anwendung des Distanzmaßes für die Clusteranalyse zur Restrukturierung gegebener Systeme auf Klassenebene, d.h. die Zusammenlegung von Klassen zu Subsystemen soll optimiert werden, und
2. Anwendung von CrocoCosmos zur Identifikation von Refactoring-werter Softwareteile (s.u.), d.h. die feingranulare, inkrementelle Überarbeitung gegebener Softwareprodukte.

Beide Fälle können im Gegensatz zum Abschnitt 9.2 relativ kurz gefaßt werden, da

- das erste Anwendungsszenario (Clusteranalyse) detailliert in [Löff99] beschrieben wird und dort mehr das Distanzmaßkonzept als die Multisicht CrocoCosmos erfolgreich validiert wird (vgl. auch [SiLö00]), und
- das zweite Anwendungsszenario (Refactoring-Identifikation) ebenfalls bereits publiziert ist ([SiStLe01]) und lediglich darauf beruht, mittels CrocoCosmos Maßnahmen zu identifizieren, die bereits empirisch validiert sind.

Für die Vollständigkeit sollen dennoch beide Arbeiten hier noch einmal kurz zusammengefaßt werden.

9.3.1 Semiautomatische Gruppierung von Klassen

Mit zunehmender Größe objektorientierter Systeme ist „[...] *die Klassenstruktur als standardmäßige Abstraktionsschicht auf Designebene oft zu feingranular und mit zu vielen für das Grundverständnis des Gesamtsystems unnötigen Details behaftet*“ ([SiLö00], S. 153). Ein wesentliches Ziel für eine bessere Verständlichkeit ist daher die Bildung einer zusätzlichen, abstrakteren Sicht auf das System, die durch *Subsysteme*, d.h. Gruppierungen von Klassen, gegeben ist. Auch wenn diese eigentlich im Zuge der Produktentwicklung entworfen werden soll, so kann es mehrere Gründe geben, die eine nachträgliche (Neu-) Gruppierung eines Produkts notwendig machen können:

- Das Produkt ist inkrementell erstellt worden und besitzt erst nach einer gewissen Zeit eine Größe, ab der eine Subsystemschicht sinnvoll ist. Diese soll dann durch Identifikation entsprechender Partitionierungen im nachhinein auf das System abgebildet werden.
- Das Produkt besitzt aufgrund des Gesetzes der zunehmenden Entropie (vgl. Kapitel 8) eine Subsystemstruktur, die nicht mehr adäquat zum aktuellen Produktstand ist, d.h. statt die Verständlichkeit zu erhöhen wird sie durch unangebrachte Subsysteme erschwert. Bei derartigen Systemen muß für jedes Subsystem überprüft werden, inwieweit es noch dem Ziel der erhöhten Verständlichkeit dient, inwieweit es vollständig (d.h. es besitzt alle Klassen, die entsprechend des gewählten Kriteriums für die Partitionierung zusammengehören) oder inwieweit es überfüllt ist (d.h. es sind zu viele Klassen im Subsystem).
- Die Gruppierung entsprechend eines vorgegebenen Kriteriums soll als eine weitere Sicht auf das System erstellt werden, soll also Teilergebnis einer Reverse-Engineering-Tätigkeit (vgl. Kapitel 3.3.3) sein. In diesem Fall soll das System nicht direkt in die neue Gruppierung überführt werden, sondern anhand ihrer besser verstanden werden.

Für den Prozeß der Gruppierung können verschiedene Kriterien angewendet werden, die wiederum zu verschiedenen Gruppierungen führen können: „*Die möglichen Kriterien können dabei nicht einfach als ‚korrekt‘ oder ‚unkorrekt‘ eingeordnet werden* [...]. *Vielmehr ist die Brauchbarkeit eines Kriteriums für ein bestimmtes Ziel von Bedeutung*“ ([SiLö00], S. 154). Mit dem in dieser Arbeit vorgestellten Distanzmaß ist ein Konzept vorgestellt, mit dem diese Kriterien formalisiert und aufgrund der Generizität des Ansatzes mit bereits fertigen Werkzeugen bearbeitet werden können.

Ziel des semiautomatischen Gruppierens von Klassen ist, nach einer Konkretisierung des Kriteriums, nach dem Klassen gruppiert werden sollen, mit möglichst wenig Interaktionsaufwand für den Anwender gute, d.h. dem Ziel der Gruppierung angemessene Subsysteme vorzuschlagen. In Anlehnung an die verschiedenen Motivationen für eine Gruppierung (s.o.) kann eine solche wie errechnet für das System angewendet werden, mit der bestehenden Gruppierung verglichen werden oder als zusätzliche Sicht zur bisherigen Gruppierung aufgefaßt werden.

Das für eine Gruppierung notwendige Procedere ist bereits durch die Prozesse zur Kohäsionsbetrachtung bekannt: Je nach Kriterium kann entweder der Prozeß zur kopplungbasierten Kohäsionsbetrachtung (vgl. Kapitel 7.3.1) oder der Prozeß zur allgemeinen Kohäsionsbestimmung (vgl. Kapitel 7.3.2) verwendet werden. Um den Prozeß soweit wie möglich zu automatisieren, wird die in beiden Prozessen aufgeführte Aktivität „Visualisierung und/oder Clusteranalyse" primär durch die Clusteranalyse bewältigt, da diese mit entsprechenden Eingabedaten eine Ergebnismenge errechnet, aus der der Entwickler geeignete wählen kann; die Anwendung der Visualisierung dient hier eher einem gesteigerten Verständnis als der konkreten Herleitung neuer Subsysteme.

Das für das folgende Beispiel verwendete Gütekriterium für eine Gruppierung kann in drei Qualitätskriterien verfeinert werden (vgl. [SiLö00], S. 164):

- Zwischen den Subsystemen soll eine möglichst geringe Interaktionskopplung bestehen,
- innerhalb eines Subsystems sollen die Klassen eine hohe Interaktionskopplung zueinander aufweisen und
- die Granularität der Subsysteme soll dem Ziel der erhöhten Verständlichkeit dienlich sein, d.h. die Anzahl der Klassen pro Subsysteme soll weder zu groß noch zu klein sein.

Diese Qualitätskriterien entsprechen direkt der „*Konstruktion der ungewichteten Interaktionskohäsion: Mehrere Klassen gehören entsprechend der ungewichteten Interaktionskohäsion um so eher zusammen, um so mehr gemeinsame Klassen (inkl. ihrer selbst) sie jeweils verwenden*" (Definition: „Ungewichtete Interaktionskohäsion", Kapitel 8.3.2). Deren Anwendung zusammen mit der in Kapitel 6.4.1 vorgestellten Technik der Clusteranalyse wird im folgenden für ein praktisches System angewendet.

Semiautomatische Gruppierung von Crocodile

Wie bereits am Anfang von Abschnitt 9 erläutert, wird für den Prozeß der Restrukturierung ein eigenes Softwareprodukt als Untersuchungsgegenstand verwendet. Es handelt sich dabei um die Meßumgebung Crocodile (vgl. Kapitel 3.5) in der Version 1.2. Wesentliche Gründe für dieses Softwareprodukt als Untersuchungsgegenstand waren:

- *Validierbarkeit*: Zu Crocodile sind umfangreiche Detailkenntnise vorhanden, die für die Beurteilung der Gruppierungsergebnisse notwendig sind.
- *Systemgröße*: Crocodile wird als Forschungswerkzeug stark inkrementell entwickelt, d.h. zu Beginn der Entwicklung konnte nicht abgesehen werden, welche Funktionalität Crocodile anbieten soll, da dieses von begleitenden Forschungsaktivitäten abhing. Die damit verbundene Entscheidung, Crocodile zu Beginn ohne eine explizite Subsystemebene zu erstellen, d.h. alle Klassen befinden sich in einem Subsystem, war in der Version 1.2 obsolet geworden: Mit 31 Klassen, die zum Teil zu stark unterschiedlichen Funktionsbereichen wie z.B. GUI oder Implementierung regulärer Ausdrücke gehören, schien die Notwendigkeit der Aufteilung in verschiedene Subsysteme offensichtlich.

Nach Berechnung der Distanzen für die ungewichtete Interaktionskohäsion, die vollständig automatisch abläuft, kann mit der Clusteranalyse begonnen werden. Wie bereits in Kapitel 6.4.1 aufgezeigt ist das Ergebnis eine Reihe von verschiedenen Partitionierungen unterschiedlicher Clusteranzahl. In Abbildung 69 sind zwei derartige Partitionierungsreihen, die jeweils mit unterschiedlichen Verfahren zur Distanzberechnung zwischen Clustern berechnet wurden, dargestellt: Im linken Diagramm wurde das Verfahren von Ward (vgl. Kapitel 6.3.1), im rechten das Median-Verfahren (vgl. [Löff99], Kapitel 4.2) angewendet:

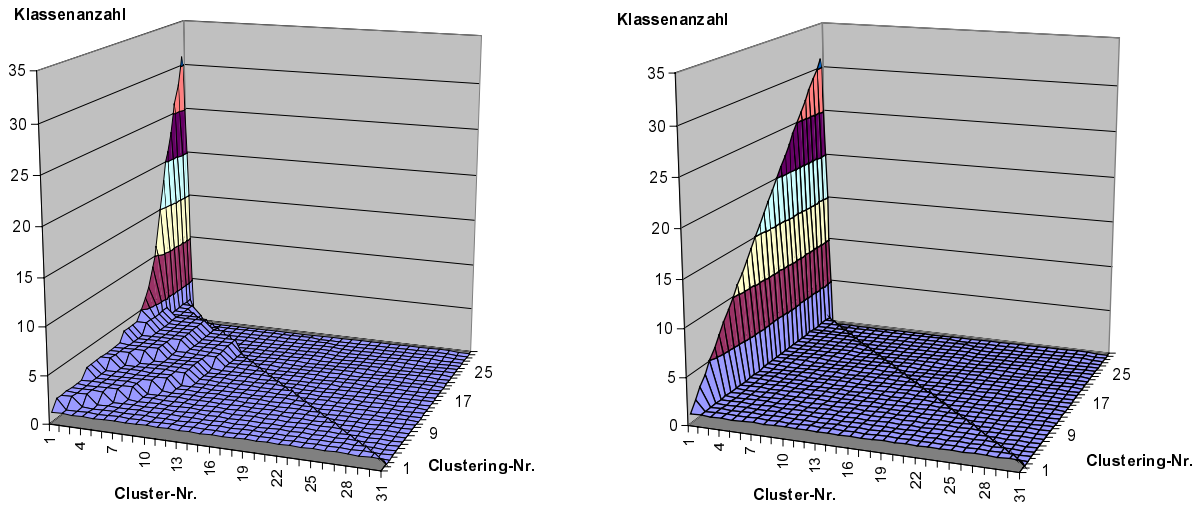


Abbildung 69: Partionierungsreihen der Clusteranalyse für das System Crocodile: Links das Verfahren von Ward, rechts das Median-Verfahren

Die Abbildungen sind dabei wie folgt zu interpretieren:

- Auf der X-Achse (Breite) sind die verschiedenen vorgeschlagenen Partitionierungen, die die späteren Subsysteme darstellen, ersichtlich: Da beide Techniken agglomerative Verfahren sind, besteht die erste Partitionierung jeweils aus 31 Clustern, die letzte Partitionierung besteht nur noch aus einem Cluster. Die Cluster sind jeweils geordnet nach Anzahl darin vorkommender Klassen (s.u.).
- Auf der Z-Achse (Tiefe) sind die Iterationen des Clusterverfahrens aufgezeichnet: Mit jedem neuen Schritt werden zwei Cluster zusammengefaßt, d.h. die Anzahl der Subsysteme (Cluster-Nr.) dekrementiert.
- Auf der Y-Achse (Höhe) ist für jede Cluster-Nr, d.h. jedes spätere Subsystem, die Anzahl der darin befindlichen Klassen angegeben. Die jeweiligen Subsysteme sind in X-Richtung jeweils nach diesem Y-Wert geordnet, damit innerhalb der Visualisierung keine Werte überdeckt werden. Zu Beginn besitzt jedes Subsystem eine Klasse, am Ende besitzt ein Subsystem alle 31 Klassen.

In [Löff99] werden Kriterien angegeben, die helfen, *sinnvolle Gruppierungen*, wie sie in diesem Fall das Verfahren von Ward leistet, von *sinnlosen Gruppierungen*, wie sie in diesem Fall z.B. das Median-Verfahren bildet, indem es während aller Partitionierungen das jeweils wegfallende Cluster genau einem Cluster zuordnet, zu unterscheiden. Im konkreten Kontext ergaben 5 von 7 Clustertechniken sinnvolle Gruppierungen (vgl. [Löff99]).

Eine wichtige Interaktion während der semiautomatischen Gruppierung von Klassen ist die Angabe der gewünschten Clusteranzahl. Folgende Aspekte können diese Entscheidung mit beeinflussen:

- Architekturwissen der Entwickler, das eine grobe Subsystemübersicht ermöglicht. Im konkreten Fall waren zumindest funktional begründbare Systemteile wie GUI, GUI-Wrapper (für eine einfachere Anpassung von Crocodile an zukünftige GUI-Frameworks) und eine Kernfunktionalität im Vorfeld bekannt. In letzterer wurde eine fertige Bibliothek wiederverwendet (*GNU-Regular Expressions*).
- Klassenverteilung innerhalb der erstellten Partitionierungen, d.h. sind die Klassen quantitativ ungefähr gleichmäßig auf die Subsysteme verteilt. Für diesen Aspekt ist das in Abbildung 69 angegebene Diagramm eine erste Hilfe: Da die Y-Werte für eine Partitionierung jeweils sortiert dargestellt werden, sind besonders solche Ergebnisse interessant, deren Höhe gleichmäßig aus X- und Z-Richtung zunimmt. Diese Bedingung ist z.B. für das Median-Verfahren nicht erfüllt, da der Y-Wert in X-Richtung stark springt (jeweils von Cluster-Nr. 1 zu Cluster-Nr. 2). Eine detailliertere Analyse dieser Werte ist durch einen Wertevergleich des Maßes P_Cl, das schließlich die Anzahl der Klassen eines Subsystems bestimmt, möglich. In Abbildung 70 sind diese Werte z.B. für alle Verfahren, die Crocodile sinnvoll in 4 Subsysteme gruppiert haben, eingezeichnet (die Reihenfolge der Cluster in X-Richtung ist hierbei ohne Bedeutung):

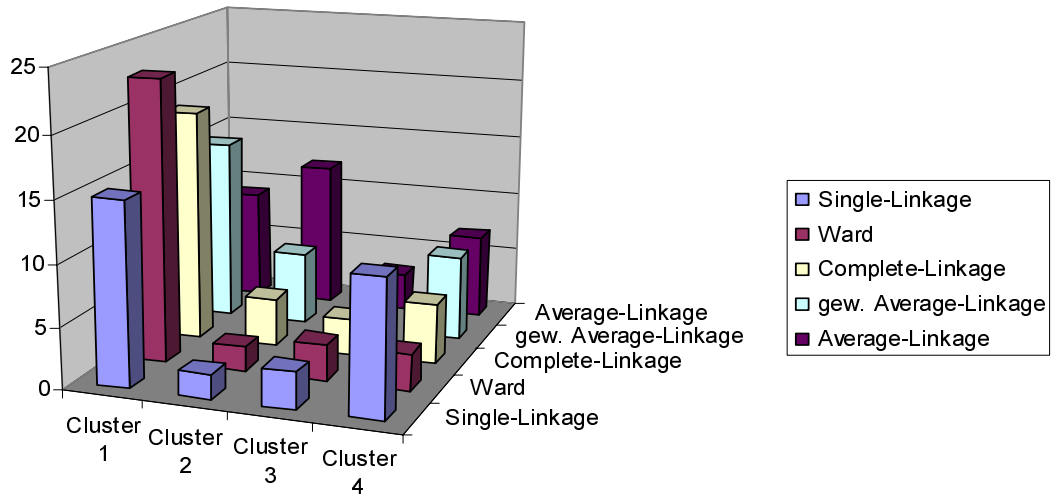


Abbildung 70: Werte des Maßes P_Cl für 5 Verfahren an der Stelle, an der Crocodile jeweils in 4 Gruppen partitioniert worden ist.

  Die einzelnen Verfahren (vgl. [Löff99], Kapitel 4) sind dabei nach der Summe der Quadratwurzeln der Klassenanzahlen der einzelnen Cluster geordnet.
- Trial-and-Error-Vorgehen, d.h. es kann mit verschiedenen Clusteranzahlen herum experimentiert werden. Dieser Aspekt wird in [Löff99] vertieft behandelt.

Für das System Crocodile wurde die Anzahl der zu erstellenden Subsysteme auf vier festgelegt, da besonders der erste Punkt, das Architekturwissen der Entwickler, diese Anzahl wünschenswert erschienen ließ.
Nach Eingabe der Clusteranzahl werden alle Ergebnisse der sinnvollen Clusterverfahren errechnet und bzgl. der oben aufgeführten Qualitätskriterien untersucht. Die Forderung nach einer bestmöglichen Granularität der entstehenden Subsysteme kann durch Beobachtung der Daten, wie sie in Abbildung 70 dargestellt sind, untersucht werden. Für die Forderung nach möglichst geringer Interaktionskopplung zwischen den entstehenden Subsystemen ist eine Betrachtung des Maßes P_EffIntCp_P sinnvoll, da hiermit die ausgehenden Interaktionskopplungen zwischen den Subsystemen betrachtet werden. Für die einzelnen in Abbildung 70 vorgeschlagenen Gruppierungen sind für Crocodile die in Abbildung 71 dargestellten Werte bestimmt worden (die einzelnen Verfahren sind dabei in Anlehnung an Abbildung 70 angeordnet):

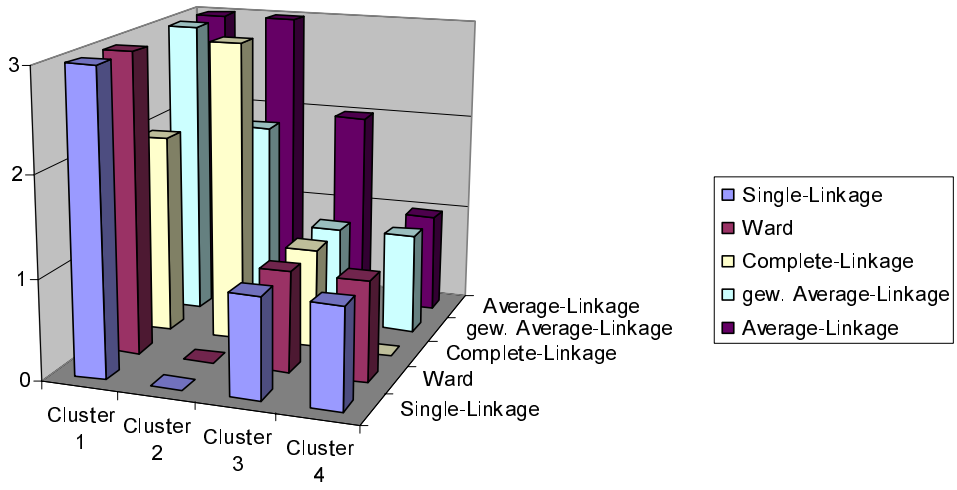


Abbildung 71: P_EffIntCp_P-Werte der Ergebnisse der 5 Clusterverfahren für Crocodile

Vor dem nächsten Schritt, der Gruppierung der Klassen von Crocodile entsprechend einer konkreten Lösung, muß der Anwender die verschiedenen Ergebnisse gegeneinander abwiegen. Im konkreten Fall war dies einfach, da das Single-Linkage-Verfahren bzgl. beider Qualitätskriterien – gute Granularität und geringe Interaktionskopplung zwischen den Subsystemen – die besten Ergebnisse erbrachte. Das konkrete Ergebnis des Single-Linkage-Verfahrens kann auch mittels eines vereinfachten Klassendiagramms dargestellt werden (vgl. linkes Diagramm in Abbildung 72) und mit einer anderen Lösung – in diesem Fall der des Average-Linkage-Verfahrens – kontrastiert werden (vgl. rechtes Diagramm in Abbildung 72).

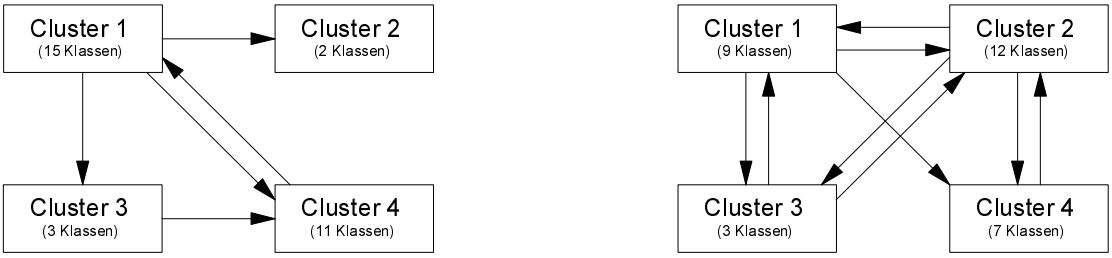


Abbildung 72: Klassendiagramm für Ergebnis des Single-Linkage (links) und Average-Linkage(rechts) Verfahrens

Das Resultat des Single-Linkage-Verfahrens wurde anschließend anhand der Crocodile-Systemkenntnisse evaluiert. Dabei konnten den errechneten Subsystemen folgende Funktionalitäten zugeordnet werden (vgl. [SiLö00], S. 167ff):

- Der Cluster 1 enthält nur GUI-Klassen, die Crocodile für die Oberflächenpräsentation benötigt.
- Der Cluster 2 enthält die aus der GNU-Regular-Expressions-Bibliothek wiederverwendeten Klassen, die von Crocodile für die Realisierung der Maße auf Dateiebene benötigt werden.
- Der Cluster 4 besteht aus zwei Funktionalitäten: Die Wrapper-Klassen der GUI und die eigentliche Funktionalität von Crocodile.
- Der Cluster 3 stellt ein spezielles, mehrstufiges Iteratorenkonzept dar, das von Crocodile's Kernfunktionalität benötigt wird.

Damit hat das semiautomatische Verfahren zur Gruppierung von Klassen aufbauend auf der ungewichteten Interaktionskohäsion eine Gruppierung vorgeschlagen, die von Systemkennern als

äußerst treffend, d.h. dem Ziel einer besseren Verständlichkeit dienlich bezeichnet wurde. In Zweifelsfragen, ob z.B. die Wrapper-Klassen, die Crocodile's Kernfunktionalität an die GUI anbinden, in ein separates Subsystem ausgelagert werden sollen, kann wiederum eine Visualisierung der Distanzen hilfreich sein: In Abbildung 73 ist eine solche, mit den jeweils errechneten Subsystemen annotierte Darstellung angegeben. Im Gegensatz zu CrocoCosmos wurde hier die Hauptkomponentenanalyse als Dimensionsreduktionstechnik (vgl. Kapitel 6.4.2) und Mathematica® als Visualisierungsfrontend verwendet. Die zusätzlich eingezeichneten vertikalen Linien dienen nur einer verstärkten Tiefenwirkung der einzelnen Punkte.

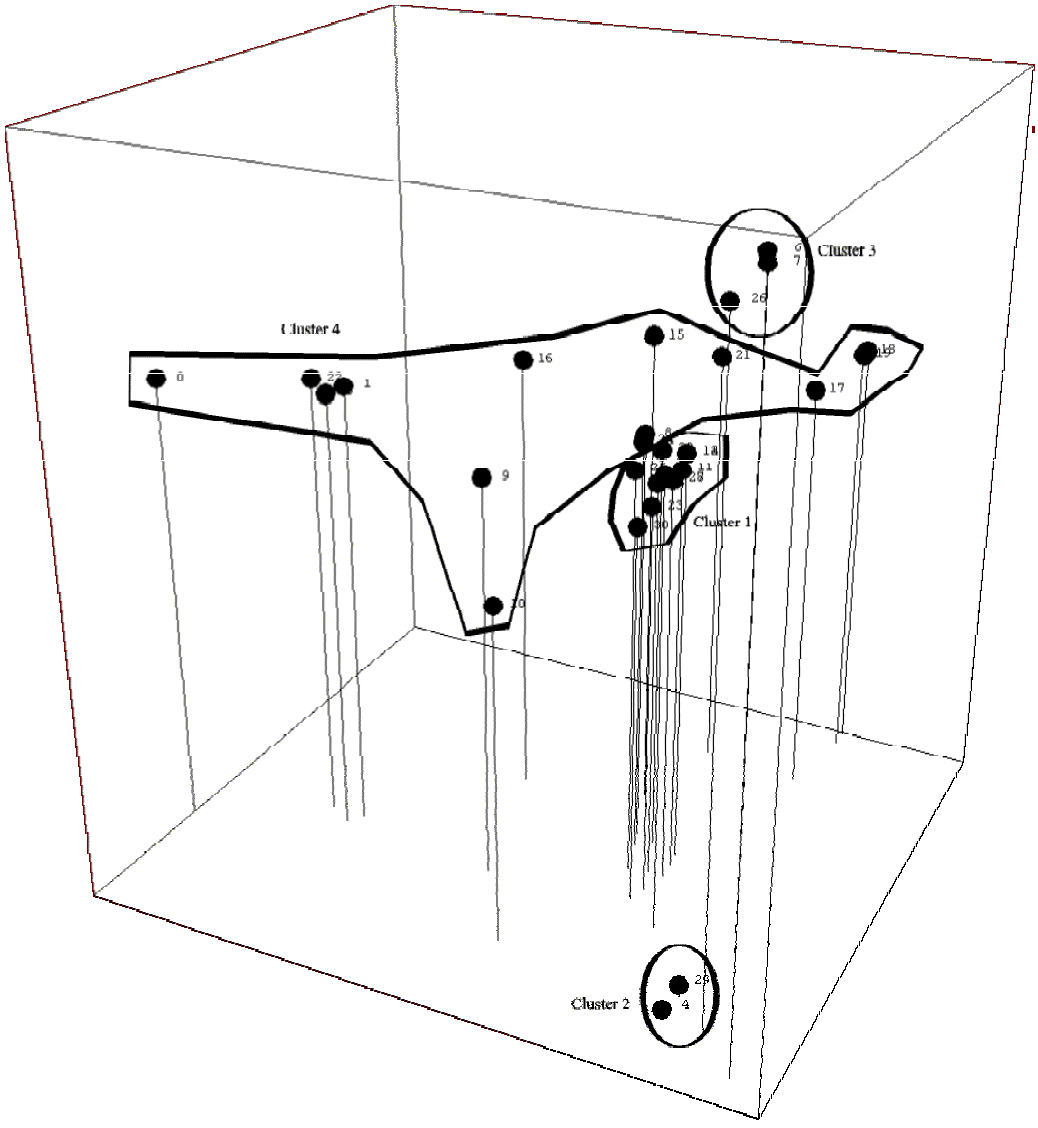


Abbildung 73: Visualisierung der ungewichteten Interaktionskohäsion für Crocodile mit annotierten Clusterergebnissen

Ähnlich positive Daten, bei denen durch minimale Interaktion des Entwicklers sehr gute Klassengruppierungen vorgeschlagen werden, sind in [Löff99] für zwei weitere, größere Projekte vorgestellt. Dort wird das gesamte hier vorgestellte Procedere ebenfalls für zwei andere Formen der Kriterien zur Gruppenbildung beschrieben: Einmal unter Anwendung der allgemeinen Vererbungskohäsion, d.h. die Gruppenbildung soll in Anlehnung an Vererbungsstrukturen erfolgen und einmal unter Anwendung der allgemeinen, merkmalbasierten Kohäsion, wobei hierfür Namensähnlichkeiten eingeführt worden sind. Letztere Variante ist besonders dann vielversprechend, wenn das zu gruppierende Produkt unter Anwendung entsprechender Namenskonventionen erstellt wurde.

9.3.2 Identifikation Refactoring-werter Softwareteile

Im letzten Abschnitt wird der gewählte Meßansatz für den Prozeß „Überarbeitung eines Programms“ validiert. Die geschieht durch die Demonstration, daß durch Anwendung des Meßansatzes – und hier insbesondere von CrocoCosmos – die Identifikation Refactoring-werter Softwareteile (s.u.), deren Nutzen für die interne Qualität von Software bewiesen ist (vgl. [Fowl99], [LuNe00]), erleichtert werden kann (vgl. [SiStLe01]).

Für das Erläutern von Refactorings ist eine Kontrastierung zum Bereich der Reengineering-Maßnahmen sinnvoll, die durch folgende Charakteristika ausgezeichnet sind (vgl. Kapitel 3.3.3):

- umfangreiche Modifikationen am Produkt inklusive der Möglichkeit der Änderung externen Verhaltens,
- sorgfältige Planung bzgl. der i.d.R. umfangreichen Ressourcen (Zeit, Personal) und
- Anwendung innerhalb später Phasen der Softwarebearbeitung (Wartung).

Refactorings sind dagegen wie folgt definiert:

> *„Refactoring (noun): a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.“*

Definition 42: Refactoring (aus [Fowl99], S. 53

Damit kontrastiert das Refactoring das Reengineering wie folgt:

- „[...] *The purpose of refactoring is to make the software easier to understand and modify.* [...] *Refactoring does not change the observable behavior of the software. The software still carries out the same function that it did before*“ ([ebd.], S. 54).
- *„All the refactorings* [...] *can be accomplished in a few minutes or an hour at most“* ([ebd.], S. 359).
- „[...] *Refactoring is not an activity you set aside time to do. Refactoring is something you do all the time in little bursts. You don't decide to refactor, you refactor because you want to do something else, and refactoring helps you do that other thing”* ([ebd.], S. 58).

Ein wesentliches Problem beim Durchführen von Refactoring-Aktivitäten ist die Frage, wann welches Refactoring an welcher Stelle im System angewendet werden soll. Fowler selbst erschwert deren Beantwortung zusätzlich durch die bewußte Verwendung unscharfer Begriffe wie „menschliche Intuition“ („*human intuition*“, [ebd.], S. 75) und „subjektive Empfindungen“ (z.B. „*smells*“ and „*stinks*“, [ebd.], S. 76).
Die externe Validierung des Meßansatzes besteht aus der Demonstration, daß das vorgestellte Meßkonzept diese Frage beantworten hilft und damit Fowlers Anmerkungen zu einer Werkzeug-unterstützung für die Identifikation Refactoring-werter Softwareteile teilweise widersprochen werden kann: „*In our experience no set of metrics rivals informed human intuition*“ [ebd.], S. 75).

Die Validierung geschieht partiell entlang des in [Fowl99] vorgestellten Refactoring-Katalogs, der aus einzelnen Refactorings besteht, die jeweils mittels einer Zusammenfassung, einer Motivation, einer Beschreibung des technischen Vorgehens und einiger Beispiele erläutert werden. Der Katalog ist in sechs Teile gegliedert (vgl. [ebd.]):

Katalogteil	Beschreibung	Refactoring-Beispiele
Zusammensetzung von Methoden („*Composing methods*“)	Sehr feingranulare Refactorings, die das Abbilden von Quelltext auf einzelne Methoden sowie die Verbesserung des Quelltextes bzgl. der Verständlichkeit behandeln.	Ersetzen einer temporären Variable durch einen Methodenaufruf („*Replace temp with query*“).
		Einführen einer erläuternden, temporären Variable („*Introduce explaining variable*“).
		Ersetzung eines Algorithmus durch einen anderen („*Substitute algorithm*“).
Objektverantwortlichkeiten („*Moving features between objects*“)	Refactorings, die die Verantwortlichkeiten von Objekten bzgl. der von ihnen angebotenen Funktionalität neu festlegen.	Verschieben einer Methode („*Move method*“).
		Verschieben eines Attributs („*Move field*“).
		Separieren einer Klasse („*Extract Class*“).
Datenorganisation („*Organizing Data*“)	Refactorings, die das Arbeiten mit Daten sowohl innerhalb eines Objekts als auch zwischen Objekten vereinfachen.	Attributsverkapselung („*Encapsulate Field*“).
		Ersetzen von Zahlen des Anwendungsbereichs durch Konstanten („*Replace magic numbers with symbolic constant*“).
		Ersetzen von versteckten Typen durch Klassen („*Replace type code with class*“).
Vereinfachung konditionaler Logik („*Simplifying conditional expressions*“)	Refactorings, die Programmteile mit konditionaler Logik vereinfachen.	Auslagerung ergebnisgleicher Bedingungen in Methoden („*Consolidate Conditional Expression*“).
		Herausfaktorisierung bedingungsunabhängiger Teile („*Consolidate Duplicate Conditional Fragments*“).
		Ersetzung von konditionaler Logik durch Polymorphie („*Replace conditional with polymorphism*“).
Vereinfachung von Interfaces („*Making method calls simpler*“)	Refactorings, die die Verwendung von Interfaces, d.h. die nach außen sichtbaren Methoden, vereinfachen.	Methodenumbenennung („*Rename method*“).
		Hinzufügen von Parametern („*Add Parameter*“).
		Verstecken von Methoden („*Hide method*“).
Generalisierung („*Dealing with Generalization*“)	Refactorings, die die Vererbungsstruktur sowie den Ort der Deklaration von Funktionalität innerhalb von Klassen aus einer Vererbungsstruktur neu strukturieren.	Herausfaktorisierung von Methoden („*Pull up method*“).
		Zuordnung von Methoden zu Unterklassen („*Push down method*“).
		Separieren einer Unterklasse („*Extract subclass*“).

Einige dieser Katalogteile sind aufgrund ihrer Granularität der zu überarbeitenden Systemteile nicht mittels des in Abschnitt 9.1 vorgestellten Werkzeugs CrocoCosmos bearbeitbar. Hierzu zählen vollständig die Teile „Zusammensetzung von Methoden“ und „Vereinfachung konditionaler Logik“, und zu großen Teilen die Bereiche „Datenorganisation“ und „Vereinfachung von Interfaces“.
Da die niedrigste Hierarchiestufe innerhalb von CrocoCosmos die Methoden und Attribute sind (vgl. Kapitel 8.1), können nur die beiden Methoden und Attribute ebenfalls atomar betrachtenden Bereiche „Objektverantwortlichkeiten“ und „Generalisierung“ damit bearbeitet werden. Nach einer Verfeinerung der Methoden in nächst kleinere Segmente kann das distanzmaßbasierte Vermessungskonzept allerdings ebenfalls auf die anderen Bereiche angewendet werden (vgl. z.B. Kapitel 7.1.1).

Im folgenden wird exemplarisch demonstriert, wie CrocoCosmos – und damit die explorierbare Visualisierung einer Kombination knoten- und kantenorientierter Maße – bei der Identifikation der zum Bereich der „Objektverantwortlichkeiten“ gehörenden Refactorings effizient eingesetzt werden kann. Betrachtet werden die Refactorings

- „Verschieben einer Methode“ („*Move Method*“, [Fowl99], S. 142ff),
- „Verschieben eines Attributs“ („*Move field*“, [ebd.], S. 146ff),
- „Separieren einer Klasse“ („*Extract class*“, [ebd.], S. 149ff),
- „Zusammenführen zweier Klassen“ („*Inline class*“, [ebd.], S. 154ff),
- „Entfernung reiner Delegationsklassen“ („*Remove middle man*“, [ebd.], S. 160ff) und
- „Verstecken durch Delegation“ („*Hide delegate*“, [ebd.], S. 157ff).

Nicht betrachtet werden die beiden Refactorings „Einfügen einer fremden Methode“ und „Einfügen lokaler Erweiterungen“, da beide im Gegensatz zu den anderen Refactorings Maßnahmen beschreiben, deren Situationen dadurch gekennzeichnet sind, daß die Probleme noch nicht implementiert sind, sondern vor der Erstellung anfallen. Beide Refactorings beschreiben Situationen, in denen das Problem darin besteht, Klassen, die nicht veränderbar sind, um Funktionalität zu erweitern. Dieses Problem besteht erst zur Entwicklungszeit, ist selbst folglich noch nicht implementiert. Da Produktmaße allerdings auf Quelltext (oder adäquaten Modellen) aufbauen, besteht das Problem der Identifikation von Situationen im Quelltext, die ein Refactoring erforderlich machen, für diese beiden Varianten nicht.

Die Beschreibung soll jeweils entlang folgenden Musters geschehen:

- Kurze Vorstellung des Refactoring,
- Charakteristika der Situationen, in denen das Refactoring notwendig erscheint,
- Portierung dieser Charakteristika auf eine angepaßte Anwendung des Prozesses zur kopplungbasierten Kohäsionsbetrachtung und
- Beispiel einer das Refactoring benötigenden Situation und deren Identifikation innerhalb von CrocoCosmos. Dies schließt die Formulierung eines geometrischen Musters ein, mit dessen Hilfe das jeweilige Refactoring motivierende Situationen erkannt werden können.

Verschieben einer Methode
Eine wesentliche Eigenschaft eines guten Entwurfs ist die korrekte Zuordnung von Methoden zu Klassen: Klassen sollten lediglich diejenigen Methoden implementieren, für die ihre Objekte später entsprechend des Verkapselungsprinzips auch verantwortlich sind. Benötigt ein solches Objekt darüber hinausgehende Funktionalität, so werden entsprechende Methoden anderer Objekte aufgerufen. Das Refactoring „Verschieben einer Methode“ besteht aus dem Verschieben der Deklaration und Implementierung einer Methode von einer Klasse zu einer anderen.

Das wichtigste Charakteristikum für dieses Refactoring, angewendet auf eine Methode m der ursprünglichen Klasse C_{alt}, ist (vgl. [Fowl99], S. 142):

- m besitzt mehr Interaktionskopplungen zu Elementen von C_{neu} als zu Elementen von C_{alt}.

Dieses kann direkt mittels der methoden- und attributbasierten Interaktionskohäsion (vgl. Kapitel 8.4) erkannt werden:

- Besitzt m mehr Interaktionskopplungen zu C_{neu} als zu C_{alt}, so äußert sich dies in der Visualisierung dadurch, daß m eine höhere Distanz zu den Elementen von C_{alt} als zu Elementen von C_{neu} besitzt. Werden die einzelnen Klassenelemente entsprechend der Klassenzugehörigkeit gefärbt, ist diese Situation sehr einfach identifizierbar.

Ein Beispiel einer solchen Situation ist bereits in Kapitels 8.4 dargestellt (inkl. des diese Visualisierung begründenden Quelltextes), in der alle Attribute (Würfel) und Methoden (Kugeln) die gleiche Größe besitzen und jeweils in Anlehnung an ihre Klassenzugehörigkeit eingefärbt sind: Deutlich ist die Methode, die für dieses Refactoring in Frage kommt, zu identifizieren (rechts unten), da sie näher zu den Elementen der neuen Klasse steht als zu den Elementen der alten Klasse.
Das für die Identifikation des Refactoring „Verschieben einer Methode" zu suchende Muster innerhalb der methoden- und attributbasierten Interaktionskohäsion beschränkt sich also auf das Suchen von Methoden (also Kugeln), die

- weit außerhalb des Bereichs der in der eigenen Klasse definierten Methoden und Attribute und
- recht dicht zum Bereich der in einer fremden Klasse definierten Methoden und Attribute liegen.

Verschieben eines Attributs
In Anlehnung an das Refactoring „Verschieben einer Methode" geht es hierbei um die korrekte Zuordnung von Attributen auf Klassen: Klassen sollten lediglich die Attribute besitzen, für die ihre Objekte auch entsprechend des Verkapselungsprinzips verantwortlich sind. Anderenfalls sollte das Attribut einer anderen Klasse zugesprochen werden (s.o.).
Das wichtigste Charakteristikum für dieses Refactoring, angewandt auf ein Attribut a der ursprünglichen Klasse C_{alt}, ist (vgl. [Fowl99], S. 146):

- a besitzt mehr eingehende Interaktionskopplungen von Elementen der Klasse C_{neu} als von Elementen der Klasse C_{alt}. Diese Interaktionskopplungen können direkt, d.h. in Form der direkten Attributsbenutzung, oder indirekt, d.h. via eigener Zugriffsmethoden geschehen („*indirect variable access*" in [Fowl99], S. 171ff). Im letzteren Fall folgt daraus die Notwendigkeit, zusätzlich zum Refactoring „Verschieben eines Attributs" das Refactoring „Verschieben einer Methode" auf diese Zugriffsmethoden anzuwenden.

Auch dieses Charakteristikum kann direkt mittels der methoden- und attributbasierten Interaktionskohäsion (vgl. Kapitel 8.4) erkannt werden:

- Besitzt a mehr Interaktionskopplungen von Elementen der Klasse C_{neu} als zu Elementen der Klasse C_{alt}, so äußert sich dies in der Visualisierung dadurch, daß a eine höhere Distanz zu den Elementen von C_{alt} als zu Elementen von C_{neu} besitzt. Werden die einzelnen Klassenelemente entsprechend der Klassenzugehörigkeit gefärbt, ist diese Situation sehr einfach identifizierbar.

Ein Beispiel einer solchen Situation ist in Abbildung 74 dargestellt[35] (für den zugrunde gelegten Quelltext s. [SiStLe01]), in der wiederum alle Attribute (Würfel) und Methoden (Kugeln) die gleiche Größe besitzen und jeweils in Anlehnung an ihre Klassenzugehörigkeit eingefärbt sind: Deutlich ist das Attribut, das für dieses Refactoring in Frage kommt, zu identifizieren (links oben: `attributeA2`), da es näher zu den Elementen der neuen Klasse steht als zu den Elementen der alten Klasse.
Das für die Identifikation des Refactoring „Verschieben eines Attributs“ zu suchende Muster entspricht demjenigen des Refactoring „Verschieben einer Methode“, nur daß hierbei Attribute zu identifizieren sind.

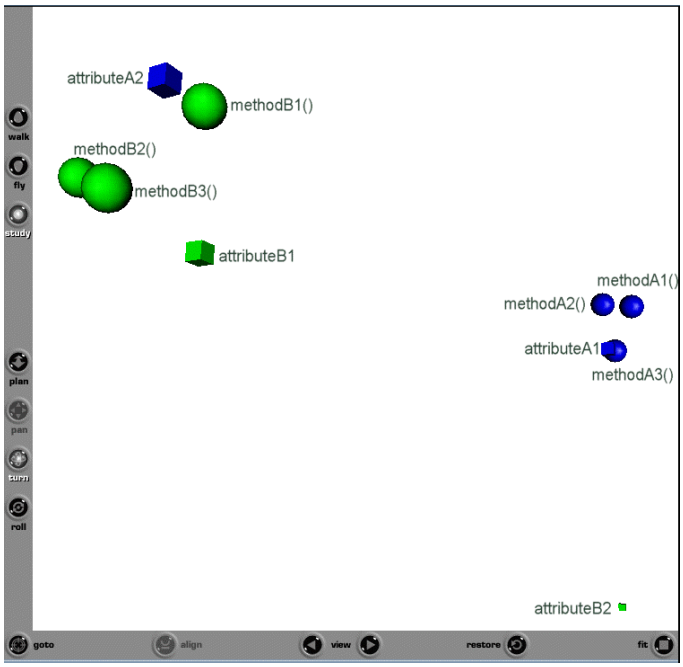


Abbildung 74(*): Beispielsituation für Refactoring "Verschieben eines Attributs"

Separieren einer Klasse / Zusammenführen zweier Klassen
Der Idealfall einer Klasse als Implementierung eines abstrakten Datentyps kann im Zuge iterativer Softwareentwicklung und damit verbundener Modifikationen für eine Klasse in zwei Extremsituationen münden: Eine Klasse ist überfrachtet, d.h. es werden mehrere abstrakte Datentypen implementiert, oder eine Klasse ist bedeutungslos, d.h. es wird nur noch marginale Funktionalität angeboten. Für den ersten Fall sieht das Refactoring „Separieren einer Klasse“ die Aufteilung der überfrachteten Klasse in separate Klassen vor; im zweiten Fall schlägt das Refactoring „Zusammenführen zweier Klassen“ das Einbinden der marginalen Funktionalität in eine andere Klasse vor. Das eine Refactoring ist die Reversion des anderen („*Inline class is the reverse of Extract Class*“, [Fowl99], S. 154).
Die wichtigsten Charakteristika für eines dieser beiden Refactorings sind (vgl. [Fowl99], S. 149 und S. 154)

- Eine Klasse besitzt sehr viele bzw. wenige Methoden und Attribute.
- Die Funktionalität einer Klasse läßt sich bzgl. der klasseninternen Interaktionskopplungen in zwei Funktionalitätsgruppen, bestehend aus Methoden und Attributen, aufteilen, bzw. die marginale Funktionalität einer Klasse besitzt hauptsächlich Interaktionskopplungen zu einer bestimmten anderen Klasse.

Diese beiden Charakteristika können wiederum direkt mittels der methoden- und attributbasierten Interaktionskohäsion (vgl. Kapitel 8.4) erkannt werden:

- Die Anzahl der Methoden und Attribute läßt sich durch bloße Abschätzung der Quantität dargestellter Objekte eines Subsystems, repräsentiert durch gleiche Färbung, sehr einfach

[35] In der dargestellten Situation kann in Anlehnung an [SiStLe01] gleichzeitig ein weiteres, hier nicht aufgeführtes Refactoring identifiziert werden: So kann das unten rechts dargestellte Attribut `attributeB2`, das offenbar von niemandem verwendet wird, ohne Probleme gelöscht werden, da es sich um ein „totes“ Attribut handelt.

erkennen. Darüber hinaus ist dieser Aspekt auch sehr effizient durch die Betrachtung der klassenkopplungbasierten Kohäsion möglich, wenn die dort verwendeten Darstellungsparameter entsprechend gewählt sind (z.B. Größe der Klassen in Anlehnung an die Anzahl der Methoden). Diese Möglichkeit wird häufig verwendet, da die Betrachtung der Klassen ein erster Schritt bei der Identifikation potentiell Refactoring-werter Softwareteile sein kann (s.u.).

- Die Identifikation von aufgrund Interaktionskopplungen gebildeter Funktionsgruppen ist Kerngedanke der methoden- und attributbasierten Interaktionskohäsion, d.h. im Falle der einfachen Möglichkeit der Aufteilung einer überfrachteten Klasse wird die konkrete Gruppierung durch die Darstellung entsprechender Cluster offensichtlich, bzw. das Zusammenführen von Klassen bereits durch eine geometrische Vermischung offensichtlich.

Eine Beispielsituation für das Refactoring „Separieren einer Klasse“ ist in Abbildung 75 dargestellt, in der wiederum alle Attribute (Würfel) und Methoden (Kugeln) die gleiche Größe besitzen und jeweils in Anlehnung an ihre Klassenzugehörigkeit eingefärbt sind (für den zugrundegelegten Quelltext s. [SiStLe01]). Deutlich ist zu sehen, daß alle dargestellten Elemente zu einer Klasse gehören, daß die Klasse eine Vielzahl von Methoden und einige Attribute besitzt, und daß sich durch die klasseninternen Interaktionskopplungen die Klasse sehr deutlich in zwei separate Teile aufteilt, innerhalb derer alle Elemente eng beieinander liegen. Jede dieser Teilfunktionalitäten stellt nach der Anwendung des Refactoring eine eigene Klasse dar.

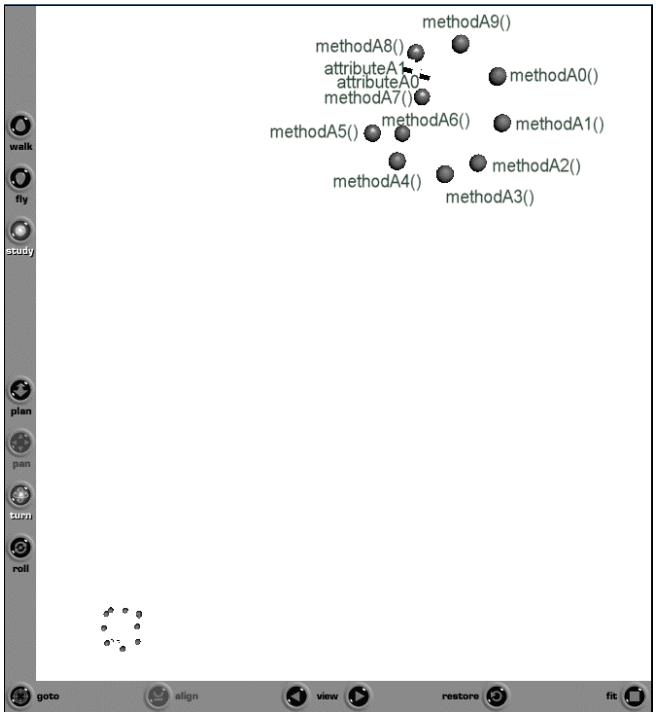


Abbildung 75[(*)]: Beispielsituation für Refactoring "Extrahieren einer Klasse"

Das für die Identifikation des Refactoring „Separieren einer Klasse“ zu suchende Muster innerhalb der methoden- und attributbasierten Interaktionskohäsion beschränkt sich also auf das Suchen von

- einer Vielzahl gleichfarbiger Methoden und Attribute und
- einer offensichtlichen, geometrischen Gruppierung zwischen gleichfarbigen Methoden und Attributen.

Eine Beispielsituation für das Refactoring „Zusammenführen zweier Klassen“ ist in Abbildung 76 mit den oben genannten Parametern dargestellt. Deutlich sind die beschriebenen Charakteristika zu erkennen: Es ist ersichtlich, daß sowohl die Elemente von zwei Klassen dargestellt sind, als auch, daß die eine Klasse eine Vielzahl von Methoden und einige Attribute besitzt, die andere dagegen nur zwei Methoden und ein Attribut. Des weiteren ist erkennbar, wie eng aufgrund der Interaktionskopplungen zwischen den Elementen beider Klassen diese Elemente beieinander liegen, d.h. die kleine Klasse ist geometrisch bereits mit der größeren vermischt.

Mit der Anwendung dieses Refactoring werden die drei Elemente der kleinen Klasse der großen Klasse zugesprochen.

Das für die Identifikation des Refactoring „Zusammenführen zweier Klassen“ zu suchende Muster innerhalb der methoden- und attributbasierten Interaktionskohäsion beschränkt sich also auf das Suchen von

- einer kleinen Zahl gleichfarbiger Methoden und Attribute und
- einer zu diesen Elementen dicht gelegenen, quantitativ nicht zu großen Gruppe andersfarbiger Methoden und Attribute, deren Farbe untereinander wieder identisch ist.

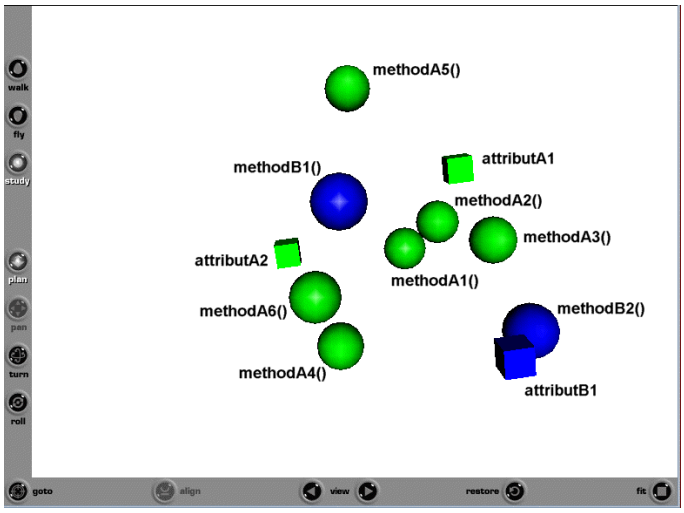


Abbildung 76[9]: Beispielsituation für Refactoring "Zusammenführen zweier Klassen"

Entfernung/Einführen von Delegationen

Eines der wesentlichen Konzepte objektorientierter Software ist das Prinzip der Verkapselung (vgl. Kapitel 8). Dies bedeutet nicht nur das Verhindern direkter Datenmanipulationen durch das Anbieten geeigneter Methoden, sondern ebenfalls die Reduktion des Umfangs der für eine Klasse bekannten Umgebung: Je weniger eine Klasse von anderen Klassen wissen muß, desto besser die Verkapselung. Das Konzept der *Delegation* unterstützt diese Forderung: Überflüssige Kenntnisse einer Client-Klasse über deren Umgebung können dadurch reduziert werden, daß diese in eine Delegationsklasse ausgelagert werden. Dies führt dazu, daß die Client-Klasse lediglich die Delegationsklasse zu kennen braucht und damit auch weniger anfällig für Änderungen der Umgebung wird, da diese durch die Delegationsklasse abgefangen werden können.
Die Delegationsklasse selbst kann entweder eine reine Delegationsklasse sein, d.h. ist nur für das Weiterleiten von Nachrichten zuständig, oder eine Mischklasse mit einer für die Delegation zuständigen Teilfunktionalität.
Der gewünschte Grad der Nutzung von Delegation in einem objektorientierten System ist sehr diskutabel: „*It's hard to figure out what the right amount of hiding is*“ ([Fowl99], S. 160) und kann während des Softwarelebenszyklus stark schwanken („*A good encapsulation six months ago may be awkward now* “ ([Fowl99], S.160).
Daher werden zwei Refactorings vorgestellt, mit denen der Grad jeweils in beide Richtungen angepaßt werden kann:

- *Einfügen von Delegationen* („*Hide Delegate*“, [Fowl99], S. 157), das mittels einer neuen Delegation Klassen verkapselt, und
- *Entfernen von Delegationen* („*Remove middle man*“, [Fowl99], S. 160), das die Delegation aufhebt und die Client-Klasse direkt mit der Service-anbietenden Server-Klasse kommunizieren läßt.

Die wichtigsten Charakteristika für das Refactoring „Einfügen von Delegationen“ zwischen einer Client-Klasse C und den von ihr aufgerufenen Server-Klassen $\mathcal{S}_i$ sind:

- Die Klasse C besitzt Interaktionskopplungen zu einigen anderen Klassen $\mathcal{S}_i$. Dadurch muß diese Klasse viel von ihrer Umgebung kennen und ist daher auch sehr anfällig für deren Änderung.
- Die Klasse C besitzt kaum eingehende Interaktionskopplungen von den Klassen $\mathcal{S}_i$, d.h. die Interaktionskopplung zwischen C und $\mathcal{S}_i$ sind i.d.R. von C ausgehend.

- Aus der Menge der verfügbaren Funktionalität der $\mathcal{S}_i$ wird jeweils nur ein sehr geringer Teil von C verwendet.

Im Gegensatz dazu sind die wichtigsten Charakteristika für das Refactoring „Entfernen von Delegationen“ einer Delegationsklasse $\mathcal{D}$ (in Rein- oder Mischform), einer Clientklasse C und den Serverklassen $\mathcal{S}_i$:

- Die Klasse $\mathcal{D}$ besitzt nur zu wenig Server-Klassen $\mathcal{S}_i$ Interaktionskopplungen. Zu den einzelnen Elementen der Server-Klassen bestehen allerdings zahlreiche ausgehende Interaktionskopplungen.
- Die Klasse $\mathcal{D}$ besitzt zahlreiche eingehende Interaktionskopplungen der Klasse C.
- Die Klasse C besitzt eine überschaubare Menge von ausgehenden Interaktionskopplungen zu anderen Klassen.

Die Charakteristika für das Refactoring „Einfügen neuer Delegationen“ lassen sich direkt auf die Visualisierung der methoden- und attributbasierten Kohäsion übertragen:

- Die Interaktionskopplungen zu einigen anderen Klassen $\mathcal{S}_i$ sind zügig durch die Menge vielfarbiger Elemente, die um die Klassenelemente von C angeordnet sind, identifizierbar.
- Die Richtung der Interaktionskopplungen ist aus der bloßen Betrachtung der Abstände nicht ersichtlich. Nach zusätzlicher Einblendung der konkreten Strukturdaten (vgl. Abschnitt 9.1) und der dort vorgenommenen Farbkodierung der Relationsrichtung ist dies allerdings sofort erkennbar.
- Daß die Klasse C von den verwendeten $\mathcal{S}_i$ nur wenig Funktionalität benötigt, ist wieder aus der bloßen Betrachtung der Distanzen der Klassenelemente ersichtlich: Während die von C verwendeten Klassenelemente sich um die sie verwendenden Teile der Klasse C anordnen, weisen die nicht von C verwendeten Klassenelemente der $\mathcal{S}_i$ von den Elementen der Klasse C weg.

Ein Beispiel einer solchen Situation ist in Abbildung 77 (annotiert) gegeben: Ausgehend von der Client-Klasse (und dort besonders von der die externe Funktionalität benötigenden Elemente in der Mitte) sind deutlich die drei verschiedenen Server-Klassen $\mathcal{S}_1$, $\mathcal{S}_2$ und $\mathcal{S}_3$ zu identifizieren. Die Server-Klassen selbst weisen allerdings explosionsartig von der Client-Klasse weg: Die Klasse $\mathcal{S}_1$ nach unten-rechts-vorne, die Klasse $\mathcal{S}_2$ nach oben-rechts-hinten und die Klasse $\mathcal{S}_3$ nach oben-links-hinten. Die wenigen jeweils dicht an Elementen der Client-Klasse C liegenden Elemente repräsentieren die wenige von C benötigte Funktionalität. In dieser Situation könnte das Refactoring „Einfügen von Delegationen“ in unterschiedlicher Weise durchgeführt werden:

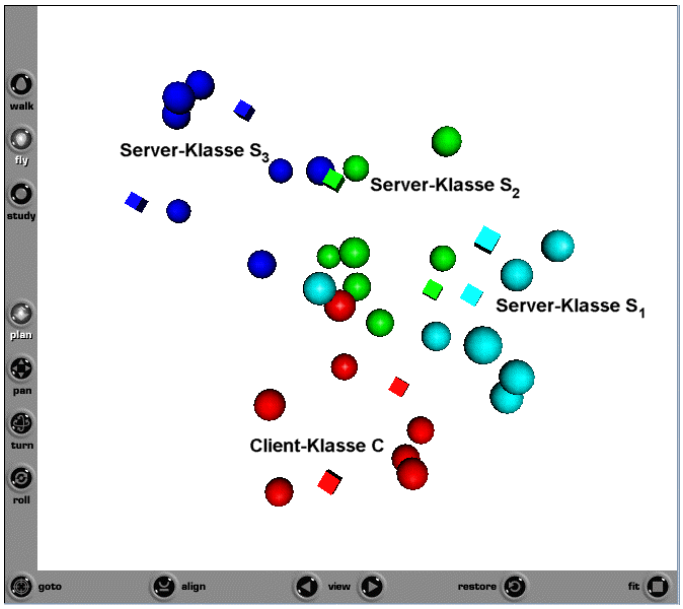


Abbildung 77[9]: Beispielsituation für Refactoring "Einfügen von Delegationen"

- Einfügen einer reinen Delegationsklasse, die die gesamte von den Klassen $\mathcal{S}_1$, $\mathcal{S}_2$ und $\mathcal{S}_3$ benötigte Funktionalität sammelt und Anfragen entsprechend delegiert.

- Einfügen von Delegationsmethoden zu einer Server-Klasse $\mathcal{S}_i$, die die Funktionalität der beiden übrigen Serverklassen sammelt und Anfragen entsprechend delegiert.
- Mischungen der beiden vorigen Varianten (z.B. Einfügen von Delegationsmethoden in $\mathcal{S}_2$, die Funktionalität an $\mathcal{S}_1$ delegiert und eine Delegationsklasse für das Verkapseln von $\mathcal{S}_2$ und $\mathcal{S}_3$.

Das für die Identifikation dieses Refactoring „Einfügen neuer Delegationen“ zu suchende Muster innerhalb der methoden- und attributbasierten Interaktionskohäsion beschränkt sich also auf das Suchen von

- mehreren relativ dicht beieinander liegenden Klassenelementgruppen, bei denen eine Gruppe jeweils einfarbig, die Gruppen untereinander aber unterschiedliche Farben aufweisen,
- einer für jede Klassenelementgruppe sternförmig nach außen weisenden Topologie und
- einem Mischbereich, in dem jeweils wenige Elemente einer Klassenelementgruppe eng beieinander liegen.

Für die Untersuchung der Richtungen der Interaktionskopplungen ist es am einfachsten, die ungewichtete Interaktionskohäsion von Klassen zu betrachten und mit den konkreten Strukturdaten anzureichern: Diese Sicht wird häufig bereits im Vorfeld für das Erkennen potentiell Refactoring-werter Klassen verwendet (s.u.).

Die Charakteristika für das Refactoring „Entfernen von Delegationen“ lassen sich ebenfalls direkt auf die Visualisierung der methoden- und attributbasierten Kohäsion übertragen:

- Die Elemente der Klasse $\mathcal{D}$ liegen eng an der von ihnen aufgerufenen Funktionalität der Klassen $\mathcal{S}_i$. Dabei existieren nicht nur punktuelle Bereiche, an denen $\mathcal{D}$ eine Nähe zu Elementen von den Klassen $\mathcal{S}_i$ aufweist (vgl. z.B. Abbildung 77), sondern es gibt eine größere Durchdringung beider Elementmengen.
- Die Klassenelemente von $\mathcal{D}$ liegen dicht an den Klassenelementen von $\mathcal{C}$. Die Richtung der zwischen den beiden Klassen liegenden Interaktionskopplungen kann nur durch Einblendung der konkreten Strukturdaten wahrgenommen werden (allerdings wird hierfür i.d.R. die Abstraktionsschicht der Klassen betrachtet, da diese auch geeignet ist für eine erste Selektion eventuell kritischer Systemteile, s.u.).
- Die Klasse $\mathcal{C}$ besitzt ein übersichtliches, nahes Umfeld von Klassenelementen aus nicht zuviel unterschiedlichen Klassen (Anzahl der Farben).

Eine Beispielsituation für das Refactoring „Entfernen einer Delegation“ ist in Abbildung 78 innerhalb der methoden- und attributbasierten Kohäsion dargestellt, deren Darstellungsparameter wie bei der Beschreibung der anderen Refactorings festgelegt wurden. Deutlich wird die Delegation als Zwischenschicht der Kommunikation zwischen Client- und Serverklasse deutlich. Dabei besitzen in diesem Fall die Elemente der Klasse $\mathcal{D}$ eine Nähe zu mehreren Elementen einer Server-Klasse, d.h. es existiert kein explosionsartiges Auseinanderdriften

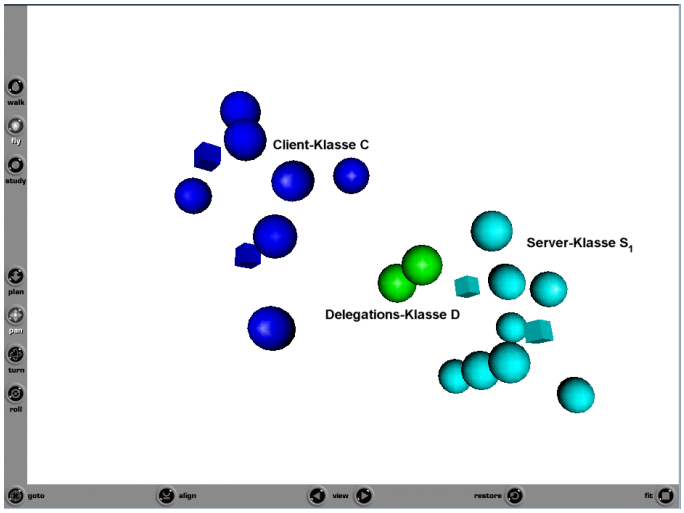


Abbildung 78(*): Beispielsituation für Refactoring "Entfernen von Delegationen"

der meisten Klassenelemente der Serverklassen von den Elementen der Delegationsklasse. Vielmehr weisen jeweils schalenartige Formen mit Klassenelementen eine gleiche Nähe zu den Elementen der Delegationsklasse auf.
Desweiteren ist die direkte Nähe der Delegationsklasse $\mathcal{D}$ zu den Elementen der Client-Klasse C zu erkennen.
Die Anwendung des Refactoring „Entfernen von Delegationen" würde in diesem Fall vermutlich das Löschen der gesamten, reinen Delegationsklasse $\mathcal{D}$, das Entfernen der Interaktionskopplungen zwischen der Client- und der Delegationsklasse und das Einfügen neuer Interaktionskopplungen zwischen der Client- und der Serverklasse bedeuten.

Das für die Identifikation dieses Refactoring „Entfernen einer Delegation" zu suchende Muster innerhalb der methoden- und attributbasierten Interaktionskohäsion beschränkt sich also auf das Suchen von

- mehreren relativ dicht beieinander liegenden Klassenelementgruppen, bei denen eine Gruppe jeweils einfarbig, die Gruppen untereinander aber unterschiedliche Farben aufweisen,
- einer ausgezeichneten Klassenelementgruppe $\mathcal{D}$, die relativ zentral zu allen betrachteten Klassenelementen liegt,
- einer in Schüsselform um $\mathcal{D}$ gelegenen Ansammlung von Klassenelementen unterschiedlicher Farbe und
- einer auf der anderen Seite der Schlüsselform gelegenen Klassenelementgruppe C, die dicht an den Klassenelementen von $\mathcal{D}$ liegen.

Die Richtungen der Interaktionskopplungen werden i.d.R. bereits im Vorfeld bei der Betrachtung der ungewichteten Interaktionskohäsion entnommen (s.u.).

Erfahrungen beim Identifizieren von Refactoring-bedürftigen Softwareteilen
Die Identifikation von Softwareteilen, bei denen die hier vorgestellten Refactorings positive Auswirkungen auf die Qualität versprechen, läßt sich sehr effizient mittels des vorgestellten Werkzeugs CrocoCosmos realisieren. Üblicherweise geschieht die Anwendung in zwei Schritten:

1. Das System wird während der Softwarebearbeitung auf der Ebene der Klassen betrachtet. Ausgehend von einem konkreten Punkt (Klasse oder Klassenmenge), an dem Funktionalität erweitert oder modifiziert werden soll, werden dabei folgende Fragen beantwortet:
   - Welches ist der nähere Kontext für die anstehende Arbeit? Hierfür wird i.d.R. eine auf Interaktionskopplung basierende Kohäsion verwendet, da hier die benutzten und benutzenden Klassen in direkter Nähe des Ausgangspunkts plaziert sind.
   - Gibt es eine Hauptrichtung innerhalb der Interaktionskopplungen? Hierfür werden entweder global für das gesamte System oder nur für ausgesuchte Teile (z.B. aufgrund des vorigen Schritts) alle Benutztrelationen (Methode benutzt Methode und Methode benutzt Attribut) eingeblendet.
   - Gibt es Vererbungsbeziehungen, die für den Ausgangspunkt relevant sind? Hierfür wird i.d.R. eine vererbungbasierte Kohäsionsbetrachtung verwendet.
   - Gibt es Extremwerte klassischer, knotenorientierter Maße für die gerade betrachtete Klassenmenge?

   Ein Beispiel einer solchen Sicht auf die Klassenebene eines realen Systems mit gleichzeitiger Einblendung aller Benutzungsrelationen, wobei die Klasse mit dem roten Kantenende eine ausgehende Interaktionskopplung zur Klasse mit dem gelben Kantenende besitzt, ist in der folgenden Abbildung 79 dargestellt (methoden- und attributbasierte Interaktionskohäsion; Größe der Klassen: C_PubM_Flat).

Bereits hier sind besondere Rollen einzelner Klassen ersichtlich: So ist z.B. die deutlich vom Rest des Systems separierte Funktionalität oben rechts identifizierbar, die ausschließlich über eine Fassaden-Klasse verwendet wird (vgl. Pfeil $\mathcal{A}$); letztere besitzt viele ausgehende Interaktionskopplungen zu dieser separaten Funktionalität, kapselt diese aber – bis auf eine Klasse (vgl. Pfeil $\mathcal{B}$) – sauber ab und wird durch eine eingehende Interaktionskopplung vom Systemkern benutzt. Die eine von der Fassadenklasse nicht vollständig gekapselte Klasse (vgl. Pfeil $\mathcal{B}$) und deren weiterer Kontext könnte im Folgeschritt für die Refactoring-Identifikation verfeinert werden.

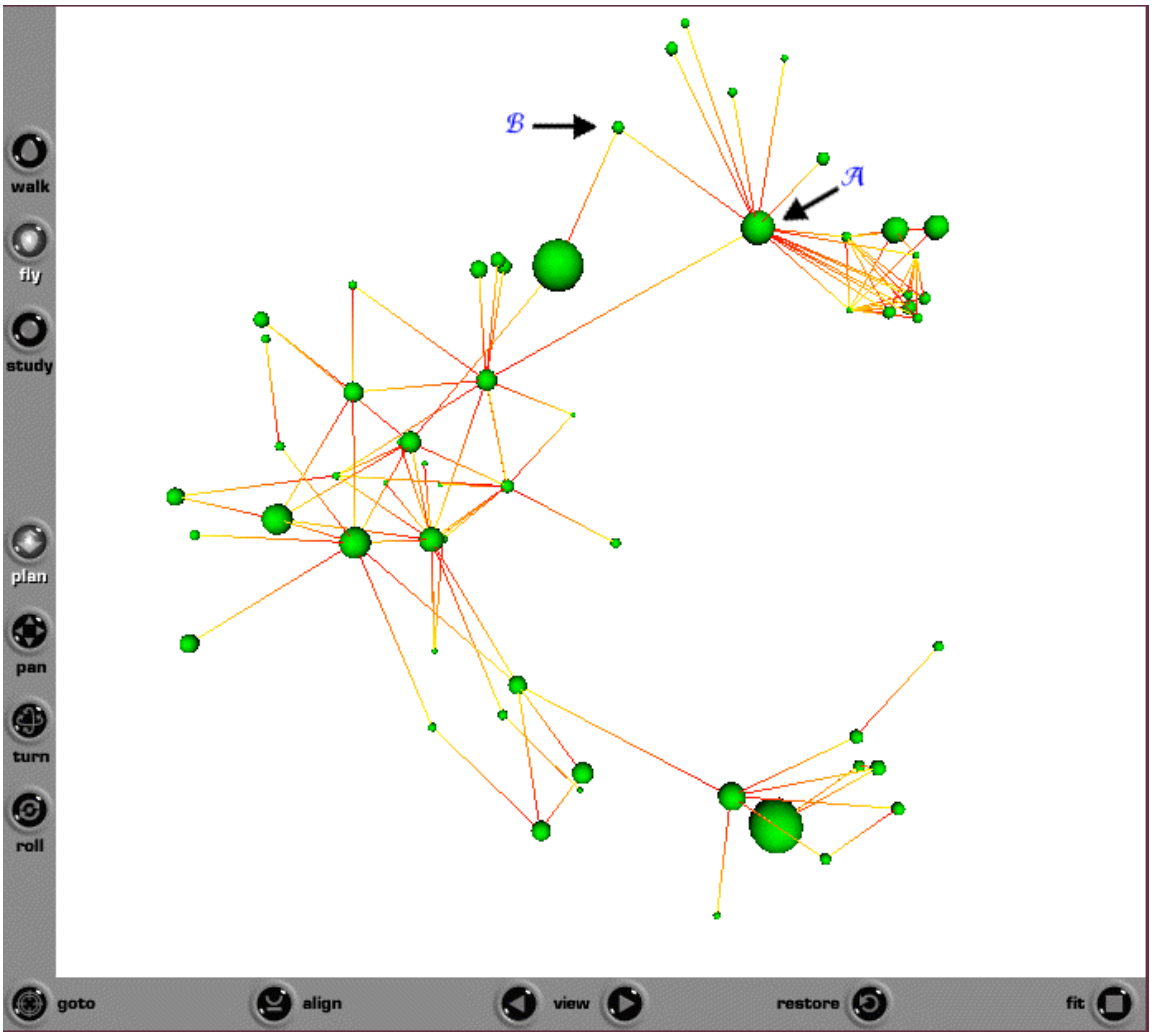


Abbildung 79(*): Beispielscreenshot einer Klassensicht auf ein System für die Identifikation weiter zu analysierender Klassen.

2. Aufbauend auf der Selektion des für den jeweiligen Arbeitsschritt relevanten Kontextes werden deren Klassenelemente mittels der methoden- und attributbasierten Interaktionskohäsion dargestellt. Dort werden dann folgende Fragen beantwortet:
   - Auf welche Klassenelemente setzt die durchzuführende Softwaremodifikation auf? Wie wird sich die Struktur verändern?
   - Treten Probleme beim ersten Schritt auf oder würden qualitative Schwachstellen durch die Softwaremodifikation auftreten? Hier setzt die Identifikation der oben vorgestellten Muster für die jeweiligen Refactorings an. Es wird jeweils überprüft, ob durch die Anwendung eines Refactoring die Softwaremodifikation einfacher durchgeführt werden könnte. Ist dies der Fall, wird das Refactoring durchgeführt.

Die Werkzeugumgebung erfordert für diese Arbeit folgende Schritte:

1. Extrahieren des Produktmodells. Hierbei können entweder die relevanten Daten für ein gesamtes Projekt oder nur für ein Teilprojekt, das durch die Verzeichnisstruktur gegeben ist, extrahiert werden. Dieser Schritt benötigt – je nach Systemgröße – einige Sekunden bis zu fast einer Stunde (z.B. für Systeme mit fast 5000 Klassen).
2. Darauf aufbauend können die klassenkopplungbasierten Kohäsionsdaten errechnet werden (vgl. Kapitel 8.3). Dieser Schritt benötigt – wiederum in Abhängigkeit von der Systemgröße – einige Sekunden bis zu einigen Stunden (z.B. für Systeme mit fast 5000 Klassen), kann allerdings, da ein nichtlineares Optimierungsverfahren eingesetzt wird, jederzeit unterbrochen werden, um mit der bis dahin errechneten Lösung weiter arbeiten zu können.
3. Für die selektierten Klassen müssen für deren Elemente entsprechend der methoden- und attributbasierten Interaktionskohäsion wiederum die Distanzen berechnet und in Koordinaten überführt werden. Da der Selektionsmechanismus i.d.R. eine sehr überschaubare Anzahl von Klassen liefert (max. 10 Klassen), nimmt dieser Arbeitsschritt nur wenige Sekunden in Anspruch.

Für die Arbeit mit CrocoCosmos zur Identifikation von Refactoring-werten Systemteilen bedeutet dies, daß die methoden- und attributbasierte Interaktionskohäsion sehr effizient für eine jeweils unterschiedliche Menge von Klassen dargestellt werden kann. Die Suche nach den oben aufgeführten Mustern für die jeweiligen Refactorings kann daher mit wenig Aufwand über verschiedene Systemteile hinweg vorgenommen werden.
Die häufig unscharfe Diskussion für oder wider ein konkretes Refactoring schlägt sich in einer ebenfalls häufig unscharfen Visualisierung nieder. Existieren klare Strukturen, die ein Refactoring dringend anraten, so werden diese Strukturen auch entsprechend klar dargestellt: Sowohl die oben dargestellten Screenshots als auch die von Fowler selbst angebrachten Beispiele sind solche eindeutigen Strukturen. Da sowohl die Erstellung der methoden- und attributbasierten Kohäsionsdarstellung als auch deren visuelle Identifikation mit wenig Aufwand durchgeführt werden kann, stellt CrocoCosmos daher ein sehr effizientes Verfahren zur semiautomatischen Erkennung von Refactoring-werten Systemteilen dar.
Darüber hinaus erlaubt CrocoCosmos die vertiefte Analyse des für viele Refactorings existentiellen Aspekts der Vererbung. Wird Vererbung, wie z.B. von Fowler, ignoriert, können sowohl falsche Refactoring identifiziert, als auch teure Folgearbeiten nach Anwenden eines Refactoring notwendig werden. Besitzen Klassen Ober- oder/und Unterklassen sind speziell die Refactorings der Kategorie „Objektverantwortlichkeiten“ erneut zu überprüfen. Hier spielt wieder das Flatten-Konzept eine wichtige Rolle: Durch die Möglichkeit, Klassen in der das Flatten vollständig berücksichtigenden Sicht (s.o.) darzustellen, werden sie auch entsprechend ihrer tatsächlichen Funktionalität analysiert. So kann z.B. eine tief im Vererbungsbaum plazierte Klasse, die nur wenige Attribute und/oder Methoden aufweist, innerhalb der normalen Sichtweise (s.o.) leicht fälschlicherweise mittels des Refactoring „Zusammenführen zweier Klassen“ in eine andere Klasse aufgelöst werden. Erst bei Berücksichtigung der ganzen, tatsächlich durch diese Klasse verfügbaren Funktionalität kann eine solche Entscheidung wirklich fundiert geschehen. Eine besondere Möglichkeit innerhalb von CrocoCosmos ist diesbezüglich die Variante, geerbte Funktionalität innerhalb einer Klasse visuell besonders hervorzuheben, um diese bei der Identifikation entsprechend berücksichtigen zu können.

Neben diesen rein strukturell begründeten Überarbeitungen, können Refactorings allerdings auch ungeachtet struktureller Eigenschaften notwendig sein, wenn spezielle, durch den Problembereich gegebene Parameter dies erfordern. Solche Fälle werden weder in [Fowl99] exemplarisch beschrieben, noch sind sie mit CrocoCosmos eindeutig identifizierbar. Hier bleibt die Intuition des Entwicklers, die sich hier besonders auf den Problembereich bezieht, dominante Kraft. In

solchen Fällen kann die Visualisierung weniger zur Identifikation als für die Erstellung eines hervorragenden *Kommunikationsmediums* verwendet werden, anhand derer die Parameter des Problembereichs besprochen werden können. Da keine Aussagen über die Verteilung strukturell bedingter und dem Problembereich entnommener Refactorings möglich sind, können ebenfalls keine Angaben zur Effektivität von CrocoCosmos gemacht werden, wieviele der notwendigen Refactorings sich mittels CrocoCosmos identifizieren lassen. Je mehr die Besonderheiten des Problembereichs auf entsprechende Entwürfe abgebildet werden, die deren Erkennen durch eine Analyse der Struktur erlauben, desto höher kann CrocoCosmos' Effektivität sein.

Ein weiterer wesentlicher Schritt der Refactorings ist die Betrachtung des Ergebnisses, d.h. die Beurteilung, ob das Refactoring die Verständlichkeit des Programms tatsächlich erhöht hat. Dies ist durch die Betrachtung des Ergebnisses mit denselben Mitteln, wie sie für die Identifikation verwendet wurden, sehr einfach möglich: So stellt sich der Programmteil, in dem zwei kleine Klassen sehr intensiv miteinander interagieren und dadurch das Refactoring „Zusammenführen zweier Klassen" motivieren, nach der Refactoringanwendung als eine bzgl. der Interaktionskopplung kohäsive Klasse dar. Entsprechend führt das Anwenden des Refactoring „Einführen neuer Delegationen" zu einer Visualisierung, in der die Client-Klasse deutlich von den Server-Klassen entkoppelt wurde.
Technisch ist dieses Feedback allerdings zur Zeit noch mit relativ viel Rechenzeit verbunden, da sich durch Anwendung des Refactoring das Produktmodell verändert hat (z.B. neue Klassenzugehörigkeit von Methoden und Attributen oder Einfügen neuer Klassen). Während bei der Identifikation Refactoring-werter Stellen der Schritt 1) und 2) nur einmal durchlaufen werden muß, um dann für beliebig viele Betrachtungen der methoden- und attributbasierten Kohäsion verwendet werden zu können, muß für jede Erzeugung des Feedbacks der Schritt 1) und 2) erneut durchgeführt werden. Modernere Mechanismen, die helfen, diejenigen Systemteile zu identifizieren, die von einer Änderung betroffen sind und dann nur für diese Teile das Produktmodell neu zu erstellen, werden allerdings auch diesen Schritt in Zukunft deutlich schneller durchführen lassen können.

9.4 Zusammenfassung

Die externe Validierung des in dieser Arbeit vorgestellten generischen Distanzmaßes vervollständigt die Untersuchung mit der Demonstration seiner Gebrauchstauglichkeit. Damit diese nicht nur für Systeme akademischen Ursprungs, sondern ebenfalls für praxisnahe Softwareentwicklungen beurteilt werden kann, ist mit CrocoCosmos eine zuverlässige und vor allen Dingen skalierende Multisicht entwickelt worden, die konkrete Instanziierungen des generischen Distanzmaßes als primäre Datenquelle für die Erstellung einer dreidimensionalen Visualisierung verwendet. Um dem die Meßtechnik erweiternden Charakter des Distanzmaßes gerecht zu werden, können die Meßwerte klassischer, knotenorientierter Maße in diese Visualisierung durch Veränderung freier Darstellungsparameter anpaßbar eingeblendet werden. Für ein kausales Verständnis der numerischen Werte während einer Analyse ist darüber hinaus die Einblendung konkreter Strukturdaten möglich.
CrocoCosmos wurde innerhalb des Prozesses „Beurteilung von Produktqualität“ für zwei große, aus der Praxis stammenden JAVA-Projekte angewendet. Für beide Projekte wurde ein Prozeß verwendet, der, aufgrund der vielen abhängigen Parameter für ein angepaßtes Qualitätsmodell, bewußt davon ausgeht, daß die Validität der Beurteilung mit zusätzlichen Iterationen zunimmt. In beiden Experimenten wurde der Prozeß daher beim ersten Mal initial durchlaufen, um durch die Evaluierung dessen Ergebnisses die Meßparameter für den zweiten durchgeführten Durchlauf entsprechend anpassen zu können. Die wesentlichen Ergebnisse beider Projekte sind:

- Das generische Distanzmaß bzw. dessen Anwendung zusammen mit klassischen, knotenbasierten Maßen innerhalb von CrocoCosmos ermöglicht eine sehr effiziente und von Systemkennern als sehr valide beurteilte Einschätzung bzgl. der Qualität großer Softwareprodukte, d.h. das verwendete Konzept ist erfolgreich extern validiert worden: Damit ist gezeigt worden, daß mittels des generischen Distanzmaßes reale Fragen an real existierende Projekte beantwortet werden können.
- Für eine qualitative Beurteilung fremder Produkte ist die Erstellung eines angepaßten Qualitätsmodells ein äußerst wichtiger und notwendiger Arbeitsschritt.
- Die Technik des Flatten ermöglicht eine vertiefte Analyse der Vererbungsstrukturen, die sich in der Praxis als sehr wichtig herausgestellt hat.
- Der Wert der Softwarevermessung nimmt zu mit dem Grad des Bewußtseins für die grundsätzlichen Problembereiche beim Einsatz neuer Techniken (vgl. Kapitel 3.6).

Für einen weiteren Prozeß, dem „Überarbeiten eines Programms“, wurde das Distanzmaß ebenfalls erfolgreich zweimal extern validiert, d.h. Fragen während der Überarbeitung von Softwareprodukten können durch die Anwendung des generischen Distanzmaßes ebenfalls beantwortet werden. Auf der einen Seite konnte gezeigt werden, daß damit die Identifikation Refactoring-werter Softwareteile erleichtert wird, d.h. die bei der Entwicklung aufkommende Frage „welches Refactoring wann an welcher Stelle angewendet werden soll“ erfolgreich beantwortet werden konnte. Dies wurde exemplarisch für die Refactorings des Bereichs „Objektverantwortlichkeiten“ des von Fowler vorgestellten Katalogs demonstriert. Auf der anderen Seite wurde das generische Distanzmaß für eine semiautomatische Klassenrestrukturierung mittels Clusteranalyse verwendet: Auch deren Ergebnisse wurden von Systemkennern als äußerst hilfreich angesehen.

Damit konnte insgesamt gezeigt werden, daß mit dem generischen Distanzmaß nach jeweils dessen kontextabhängigen Instanziierung eine validierte und praxisrelevante Erweiterung bisheriger Meßansätze vorhanden ist, die unter Verwendung entsprechender Werkzeuge wie z.B. CrocoCosmos effizient in die professionelle Softwareerstellung integriert und dort eingesetzt werden kann.

„Der Gelehrte vergesse,
was er getan hat,
sobald es getan ist,
und denke stets nur an das,
was er noch zu tun hat.“
(Johann Gottlieb Fichte in
„Grund unseres Glaubens“)

10 Zusammenfassung/Ausblick

In dieser Arbeit wurde gezeigt, wie die Anwendung von Instanziierungen eines neuen generischen Distanzmaßes die Möglichkeiten einer meßbasierten Qualitätssicherung großer Softwaresysteme praxisrelevant erweitert. Dies wird noch dadurch verstärkt, daß der Ansatz als Erweiterung bisheriger Meßansätze die Einbeziehung „klassischer“, knotenorientierter Maße ermöglicht.
Um die Qualitätssicherung auf diese Art und Weise betreiben zu können, ist es unverzichtbar, die zu sichernde Eigenschaft explizit zu formulieren. Die für diesen Zweck definierten Qualitätsmodelle lassen sich aufgrund der Vielzahl der auf sie wirkenden Einflußfaktoren innerhalb des konkreten Kontextes nur schwer über verschiedene Anwendungen hinweg wiederverwenden, so daß in der Praxis statt der Modelle selbst deren Erstellungsprozesse im Vordergrund stehen.
Entsprechend eines solchen angepaßten Qualitätsmodells ermöglichen Softwareproduktmaße nun die qualitative Bewertung von Softwareprodukten und ermöglichen damit das Bilden von Feedback Loops, um qualitätssicherende Maßnahmen als eine alle Phasen der Softwarebearbeitung begleitende Tätigkeit anzuwenden. Diese Feedback Loops selbst können ebenfalls für die zum Produkt führenden Prozesse und für deren Durchführung notwendigen Ressourcen verwendet werden, so daß den Produktmaßen innerhalb der Software-Technik eine besonders wichtige Rolle zukommt. Für deren praxisrelevanten Einsatz verlangen alle konkreten Anwendungsszenarien einer Softwarevermessung nach einer möglichst automatisierten und an das konkrete Qualitätsmodell anpaßbaren Werkzeugumgebung für die Produktvermessung. Sowohl diesbezügliche Defizite als auch allgemeine Probleme beim Einsatz neuer Techniken innerhalb der Softwarebearbeitung haben der Softwarevermessung allerdings bis heute keine adäquate Praxisrelevanz ermöglicht.
Eine Vielzahl der Probleme liegt in der suggerierten Erfahrung bzgl. des Einsatzes von Maßen aufgrund deren tagtäglichen Verwendung; während dort allerdings i.d.R. bereits langjährig etablierte und tradierte Meßtechniken angewendet werden, muß diese Sicherheit für den Bereich der Softwarevermessung erst noch gewonnen werden. Ein wichtiger Schritt dafür ist das Explizieren der beim täglichen Messen vorgenommenen, impliziten Annahmen und deren anschließende Übertragung auf die Softwarevermessung in Form einer Meßtheorie. Wesentliche Ergebnisse davon sind die Notwendigkeit von vollständigen, eindeutigen, konsistenten und verständlichen Produktmodellen und den darauf identifizierten, konsensfähigen Relationen. Letztere müssen im Vorfeld einer Vermessung bekannt sein, da Messen lediglich eine Automatisierung der Relationsbestimmung darstellt und darüber hinausgehende Aussagen unzulässig sind. Für ein Maß folgt daraus die Notwendigkeit der internen Validierung als eine Bestätigung seiner ausschließlich auf Aussagenerhaltung bedachten Eigenschaften und die Notwendigkeit einer externen Validierung für die Demonstration seiner Gebrauchstauglichkeit. Beide Validierungsarten werden jeweils durch einen Prozeß unterstützt.
Der Transfer der Meßtheorie auf bisherige Softwaremaße ermöglicht die Bildung eines Graphenmodells, das alle bisherigen Größen-, Kopplungs- und Kohäsionsmessungen als Knotenattributierung subsumiert und gleichzeitig Ursachen vieler Probleme beim Einsatz von Meßwerkzeu-

gen analysieren hilft. Sowohl die Probleme der Knotenlastigkeit der Meßwerkzeuge als auch der innerhalb des Graphenmodells heraus gearbeitete Freiraum der Vermessung von Kanten stellen die Grundmotivation für eine Distanzbestimmung zwischen Softwareentitäten dar.
Für die Konkretisierung dieser Distanzen werden Anleihen aus der Psychologie und der dort etablierten Gedächtnismodelle genommen, um anschließend ein allgemeines, auf gemeinsame Merkmale basierendes Ähnlichkeitskonzept zu erstellen, das einerseits einfach, universell und intuitiv ist, andererseits aber die formalen Bedingungen einer Metrik erfüllt. Die durch diese Metriken mögliche Kantenattributierung stellt eine wirkliche Erweiterung bisheriger Meßansätze dar und ermöglicht dadurch neue Techniken wie Clusteranalysen oder aussagenstarke Visualisierungen.
Diese neue Art von Maßen erlaubt ebenfalls die Nachbildung bisheriger Kohäsionsmaße, die durch den Transfer auf die Metriken und die damit verbundenen neuen Techniken dann aber einfacher zu klassifizieren, zu interpretieren (inkl. einer vereinfachten Herleitung konstruktiver Schritte) und teilweise sogar zu falsifizieren sind. Das Konzept der Kopplung, das bis jetzt eine von Kohäsion separat zu betrachtende Eigenschaft von Softwareentitäten darstellte, kann hierbei als Kohäsion stiftende Kraft aufgefaßt werden und ermöglicht ein Modell zur kopplungbasierten Kohäsionsbestimmung, das die Analyse von über bisherige Kohäsionsmaße hinausgehenden Aspekten ermöglicht. Sowohl diese Art der Anwendung der Metriken als auch die allgemeine, merkmalbasierte Distanzbestimmung wird jeweils durch einen detaillierten, der Meßtheorie gerecht werdenden Prozeß unterstützt.
Für konkrete Anwendungen dieser Prozesse bei der Kohäsionsbestimmung objektorientierter Systeme muß das Konzept der Vererbung detailliert behandelt werden, da die separate, den Vererbungskontext einer Klasse nicht berücksichtigende Betrachtung einer Klasse sowohl für bisherige Maße als auch für die Metriken häufig unvollständige Ergebnisse liefert. Die notwendige Technik des Flatten ermöglicht dann die Anwendung des Prozesses zur kopplungbasierten Kohäsionsbestimmung, deren Ergebnis mehrere, jeweils verschiedene Kohäsionsarten untersuchende und auf jeweils unterschiedlichen Kopplungsarten basierende Metriken sind. Neben der zusätzlichen Instanziierung dieses Prozesses auf der Ebene der Attribute und Methoden werden typische Ergebnisse des Prozesses zur allgemeinen, merkmalbasierten Distanzbestimmung vorgestellt.
Um die erarbeiteten, konkreten Metriken extern zu validieren, muß der Ansatz auf Projekte aus der Praxis angewendet werden, was aufgrund deren Größe wiederum eine zuverlässige Werkzeugumgebung erfordert. CrocoCosmos stellt eine derartige Umgebung dar und kombiniert bisherige, knotenorientierte Maße mit den neuen Metriken in der Erstellung dreidimensionaler, explorierbarer, virtueller Informationsräume. Zusätzliche Peripherie, wie Komponenten zur Clusteranalyse oder der statistischen Auswertung der Meßdaten, ermöglicht anschließend die externe Validierung innerhalb unterschiedlicher Prozesse der Qualitätssicherung großer Softwaresysteme. Das Validierungsergebnis belegt, daß die Metriken zusammen mit den Maßen eine sehr effiziente Möglichkeit der Qualitätsbewertung großer, objektorientierter Softwaresysteme darstellen, und daß es mit ihnen semiautomatisch möglich ist, restrukturierungswerte Softwareteile zu identifizieren und konstruktive Maßnahmen abzuleiten.

Durch die vorgestellte konzeptionelle Erweiterung klassischer, knotenorientierter und durch einen eigenen Prozeß herleitbare Softwarevermessung um Metriken, die explizit den Forderungen der Meßtheorie entsprechen, durch die bewußte Konzentration auf weitestgehende Anpaßbarkeit und Automatisierbarkeit, durch das Aufzeigen einer entsprechenden, skalierenden Werkzeugunterstützung und durch die erfolgreichen, empirischen Anwendungen des Ansatzes auf Projekte der Praxis (von denen in dieser Arbeit nur einige wenige genannt sind) ist ein wichtiger Meilenstein für die zukünftige Etablierung von Softwarevermessung innerhalb der Qualitätssicherung von Softwaresystemen geschaffen.

Die in dieser Arbeit aufgezeigten Konzepte können in Zukunft noch weiter ausgebaut werden. Folgende Bereiche sind diesbezüglich besonders relevant:

- *Produktmodelle komponentenbasierter Software*: Die Produktmodelle von Meßwerkzeugen für objektorientierte Systeme konzentrieren sich auf die klassischen Entitäten der objektorientierten Analyse und des objektorientierten Entwurfs eines Einzelplatzsystems wie z.B. Subsysteme und Klassen (vgl. z.B. Produktmodelle von Datrix und Crocodile in Kapitel 3.4.4 und Kapitel 5). Heutige Softwaresysteme verwenden aber in zunehmendem Maße verteilte Ressourcen (z.B. *Remote Method Invocation*, *RMI*) bis hin zu fertigen, vollständig gekapselten Komponenten (z.B. *Enterprise JavaBeans*, *EJB*). Während derartige Systemteile für die in dieser Arbeit vorgestellten Meßansätze ausgeschlossen werden mußten (vgl. Kapitel 9.2.4) und damit für die meßbasierte Qualitätssicherung nicht zur Verfügung standen, wird sich mit zunehmender Verwendung dieser Techniken die Komplexität dieser Systemteile und damit eben die Notwendigkeit einer meßbasierten Qualitätssicherung erhöhen. Mit der Einbeziehung komponentenbasierter Software ergeben sich folgende Fragestellungen:
  - Wie sieht ein Qualitätsmodell komponentenbasierter Software aus? Auch hier werden wieder verschiedene Einflußfaktoren jeweils unterschiedliche Sichtweisen besitzen (vgl. Kapitel 2.2), so daß die in Kapitel 2 aufgezeigten Prozesse angewendet werden müssen.
  - Wie sehen Produktmodelle komponentenbasierter Software aus, damit die in den angepaßten Qualitätsmodellen identifizierten, feingranularen Eigenschaften meßbar werden?
  - Welche Kopplungen können zwischen der Anwendung und den Komponenten und zwischen den Komponenten selbst identifiziert werden? So ist die Anwendung durch die Verwendung eines JavaBeans, das lokal implementiert, dessen Ort bekannt und dessen Funktionalität auf Signaturebene bekannt sein muß, sicherlich enger mit diesem JavaBean gekoppelt als eine Anwendung mit einem Enterprise JavaBean-Server, der sowohl Ort als auch Implementierung der verwendeten Komponente vollständig kapselt.
  - Wie sehen auf diesen Kopplungen basierende, hilfreiche Kohäsionsbetrachtungen aus?
  - Wie sehen allgemeine, merkmalbasierte Kohäsionsbetrachtungen aus (z.B. können die verwendeten Komponenten als Merkmale aufgefaßt werden, so daß in der daraus folgenden Kohäsionsbetrachtung diejenigen Softwareteile zusammengehören, die eine ähnliche Menge von Komponenten verwenden).
- *Erweiterung statischer Strukturen um dynamische Daten*: Während der Fokus dieser Arbeit auf der Qualitätssicherung der statischen Struktur großer Softwaresysteme liegt, ist ein weiteres wichtiges Qualitätsmerkmal das *Laufzeitverhalten* des Systems. Im Gegensatz zur statischen Analyse, bei der die Meßwerte lediglich vom System selbst abhängen, hängt das Laufzeitverhalten i.d.R. zusätzlich von den zu betrachtenden Anwendungsfällen ab, d.h. die für das Vermessen notwendigen Daten müssen während der Programmabarbeitung repräsentativer Anwendungsszenarien gesammelt werden. Die dafür notwendige Technik wird als *Profiling* bezeichnet und „[...] *refers to using information collected about the dynamic behavior of a program to improve optimization of that program*“ [Schm et al.98]. Von den verschiedenen bekannten Profiling-Techniken *Sampling*, d.h. das in regelmäßigen Abständen stattfindende Beobachten und Aufzeichnen relevanter Daten, dem *Tracing*, d.h. das Aufzeichnen der durchgeführten Anweisungsfolgen mit anschließender Reduktion der Daten auf Funktionsaufrufe und Bedingungsauswertungen und dem *Instrumentalisieren*, d.h. das Einfügen spezieller Anweisungen in den zu vermessenden Quelltext, die zur Laufzeit des modifizierten Systems eine Profiling-Umgebung anfüllen, gewinnt die letzte Technik zunehmend an Bedeutung (vgl. [ebd.]). Speziell JAVA-Systeme lassen sich durch das *JAVA Virtual Machine Profiler Interface* damit einfach dynamisch vermessen, da dieses Standardschnittstellen zur Verfügung stellt, mit denen die Laufzeitdaten des

instrumentalisierten Quelltextes einfach verfügbar sind (vgl. [ViLi00]). Nach dem Profiling können die für repräsentative Anwendungsszenarien erhaltenen Daten als zusätzliche Meßwerte die statische Analyse unterstützen. Möglich sind damit u.a.:

- Aussagen über *tatsächliche Kopplungen* zwischen Klassen insbesondere aus Vererbungshierarchien: Während die Flatten-Technik (vgl. Kapitel 8.2.2) lediglich in der Lage ist, potentielle Interaktionskopplungen zu berücksichtigen, um darauf aufbauend spezielle Kopplungsmaße zu bestimmen, können dann tatsächliche, allerdings von konkreten Anwendungsszenarien abhängige Kopplungsmeßwerte ermittelt werden.
- *Gewichtete Interaktionen*: Während die statische Analyse lediglich betrachten kann, ob eine Interaktionskopplung vorliegt oder nicht, können dann einzelne Interaktionen mit der Aufrufhäufigkeit gewichtet werden. In einer darauf aufbauenden Kohäsionsbetrachtung würden dann Systemteile, die bzgl. des einen untersuchten Anwendungsfalls tatsächlich häufig direkt oder indirekt interagieren, zusammengehören.
- Entdeckung *toten Codes*: Innerhalb der statischen Analyse wird lediglich untersucht, welche Methoden oder Attribute eine Methode verwenden kann. Weder die nähere statische Umgebung einer eine solche Verwendung kennzeichnenden Anweisung wie z.B. notwendige Bedingungen noch genauere Analysen eines eventuell polymorphen Aufrufs werden vorgenommen. Mit der dynamischen Analyse ist es möglich, Attribute und Methoden, die zwar dem Quelltext nach verwendet werden, dies zur Laufzeit aber nicht relevant ist, zu identifizieren. Ein Trivial-Beispiel eines solchen Falls ist die Bedingung `if (`<Bedingung>`) method1 else method2()`. Während die statische Analyse eine Verwendung der Methode `method2()` identifiziert, wird durch die dynamische Analyse die Auswertung der <Bedingung> möglich. Gilt z.B. für alle möglichen Anwendungsfälle, daß die Bedingung `false` zurückliefert, so kann der `else`-Zweig (und damit eventuell die Methode `method2()`) als für diese Anwendungsfälle toter Code identifiziert werden.

- *Erweiterung des empirischen Relationensystems:* Ein angepaßtes Qualitätsmodell mit den maximal verfeinerten Qualitätsmerkmalen und den sie bestimmenden Softwaremaßen besteht häufig aus konsensfähigen Setzungen (z.B. *keine Klasse darf mehr als 25 öffentliche Methoden besitzen*) und konsensfähigen, ordinalen Aussagen (z.B. *je mehr direkte oder indirekte Superklassen eine Klasse besitzt, desto schwieriger ist ihre Wartung*). In beiden Fällen fehlt i.d.R. entsprechende Erfahrung, was typische Meßwerte anderer, vergleichbarer Systeme sind: Für die Setzungen würde ein derartiges Wissen die häufig vorgenommene Willkür des Wertewunsches durch eine auf Vergleichen mit ähnlichen Produkten basierende Wertediskussion ersetzen können. Die ordinalen Aussagen könnten, wenn die Meßwerte der anderen Produkte mit externen Merkmalen (z.B. Wartungskosten, vgl. Kapitel 3.2) angereichert würden, durch Aussagen auf höheren Skalenniveaus (z.B. Rationalskala) erweitert werden. Mit den in dieser Arbeit vorgestellten Konzepten und Werkzeugen ist die automatische Vermessung von großen Softwareprodukten mit wenig Aufwand durchführbar und damit die Erstellung einer solchen Produktmeßdatenbank (in Anlehnung an die Prozeß- und Ressourcendatenbank von SLIM-Metrics, vgl. Kapitel 3.4.2) möglich. Diese könnte dann u.a. zur Beantwortung folgender Fragen beitragen:
  - Was sind „gute“ bzw. „schlechte“ Meßwerte für bestimmte Maße (inkl. kombinierter Maße) unter der Vorgabe bestimmter Qualitätsziele?
  - Was sind mögliche Kategorien von Software, von den verschiedenen Einflußfaktoren auf das Qualitätsverständnis und von weiteren Projektparametern (wie z.B. der verwendete Prozeß), so daß diese Kategorien für eine Kategorisierung der verwendeten Qualitätsmodelle verwendet werden können?
  - Existieren in den jeweilig identifizierten Kategorien Korrelationen zwischen bestimmten Produkt-, Ressourcen und Prozeßmaßen?

- Wie sehen auf der konstruktiven Seite sinnvolle, den jeweils verwendeten Kontext maximal berücksichtigende Programmierrichtlinien aus?

- *Semiautomatische Trendanalyse*: Der in Kapitel 3.3.4 vorgestellte Prozeß der Trendanalyse für die Anwendung von Produktmaßen auf verschiedene Versionen einer Softwareentität läßt sich innerhalb der bisherigen Werkzeugumgebung nur semiautomatisch bzw. nur für automatisch erkennbare Versionen einer Softwareentität durchführen (vgl. Kapitel 3.3.4). Das durch Umbenennung, Verschiebung oder auch Zusammenlegung mit anderen Softwareentitäten entstehende Hauptproblem der automatischen Verfolgung von Softwareentitäten über verschiedene Versionen kann durch das Auslesen entsprechender Daten des verwendeten *Konfigurations-Managements* (*Configuration Management*, *CM*) je nach dessen Ausprägung größtenteils behoben werden. Dessen Grundfunktionen lassen sich üblicherweise zusammenfassen in (vgl. [Alde93], [Dumk00])
  - *Konfigurationsidentifikation* (*Configuration Identification*), d.h. dem Identifizieren einzelner, versionierter Elemente, sogenannter *Konfigurationselemente* (*Configuration Items*),
  - *Konfigurationssteuerung* (*Configuration Control*), d.h. dem Verfolgen von durch Modifikationen einzelner Konfigurationselemente notwendigen Aktivitäten,
  - *Konfigurationsüberwachung* (*Configuration Tracking*), d.h. der Sicherstellung der Konsistenz und Vollständigkeit durchgeführter Änderungen, und in
  - *Konfigurationsprotokollierung* (*Configuration Protocolling*), d.h. dem Erfassen und Verwalten der Konfigurationselemente.

  Speziell der vom jeweils verwendeten CM-System angebotene Grad der Unterstützung der Teilfunktionalitäten Konfigurationsidentifikation und Protokollierung entscheidet dabei über den Grad der Automatisierbarkeit und der Granularität einer Trendanalyse:
  - Unterstützte Abstraktionsniveaus der Konfigurationsidentifikation: Häufig bauen CM-Systeme auf Dateiebene auf (z.B. SCCS, vgl. [Alde93]), d.h. es kann lediglich festgestellt werden, ob Dateien modifiziert wurden. Bei Verwendung entsprechender Programmier-richtlinien (vgl. Kapitel 8.5) können diese Daten jedoch problemlos auf die Klassenebene übertragen werden. Für die Betrachtung einzelner Methoden über verschiedene Versionen hinweg ist allerdings eine feingranularere Konfigurationsidentifikation notwendig.
  - Formalisierung der Konfigurationsprotokollierung: Mit zunehmendem Grad der Formalisierung der Eingabe von Modifikationen einzelner Konfigurationen (z.B. Umbenennen einzelner Versionen) können diese Daten automatisch ausgewertet werden und damit die entsprechenden, modifizierten Konfigurationselemente bzgl. der Trendanalyse untersucht werden.

  Mit einer durch das CM-System unterstützten Trendanalyse sind vor allen Dingen folgende Aspekte untersuchenswert:
  - Abhängigkeit zwischen Prozeß und Produkt: Wie wirken sich Modifikationen des verwendeten Prozesses auf die Produktqualität aus? Wird z.B. ab einer bestimmten Produktversion ein neuer Prozeß innerhalb des Unternehmens verwendet, so können eventuelle qualitative Auswirkungen in der Trendanalyse erkannt und damit Rückschlüsse auf den Nutzen der Prozeßänderung gezogen werden.
  - Abhängigkeit zwischen Ressource und Produkt: Wie wirken sich Änderungen der für die Produkterstellung eingesetzten Ressourcen auf die Produktqualität aus? Wird z.B. ab einer bestimmten Produktversion ein geändertes CASE-Werkzeug verwendet, so können eventuelle qualitative Auswirkungen in der Trendanalyse erkannt und damit Rückschlüsse auf den Nutzen der Ressourcenänderung gezogen werden.
  - Feststellung der Produktrobustheit: Wann ist ein Produkt auslieferungsfähig? Ein wichtiges Indiz dafür ist die Anzahl gefundener Fehler, die für Softwareprodukte wie folgt charakterisiert werden kann: „*For software, the error rate is at the highest level at integration and test. As it is tested, errors are identified and removed. This removal continues at a*

slower rate during its operational use; the number of errors is continually decreasing [...]" ([RoHaSh98], S. 2). Da jede Entfernung von Fehlern i.d.R. eine Softwareänderung zur Folge hat und diese – je nach Auswirkung – meßbar ist, kann eine Trendanalyse für die Entscheidung verwendet werden, wann ein Produkt auslieferungsfähig ist: Eine zunehmende Stabilität des Systems, d.h. eine geringere Modifikationshäufigkeit, läßt sich in einer zunehmenden Stabilität der Meßwerte identifizieren. Je weniger Änderungen pro Zeiteinheit durchgeführt werden, desto eher kommt eine Produktauslieferung in Frage.

- *Updatemechanismen für Produktmodellelemente und deren Meßwerte*: Sowohl für die Trendanalyse, bei der verschiedene Versionen eines Softwareprodukts jeweils neu vermessen werden, als auch für auf Softwareänderungen basierenden Feedback-Loops, bei denen die Auswirkungen durchgeführter Änderungen meßbasiert demonstriert werden sollen, sind Updatemechanismen für die Produktmodelle und die darauf aufbauenden Meßwerte sinnvoll, um speziell bei größeren Systemen die Rechenzeit zu minimieren. Ähnliche Techniken werden z.B. bei CM-Systemen in Form der Bestimmung sogenannter *Deltas* zwischen verschiedenen Versionen zur Minimierung des benötigten Speicherbedarfs angewendet (vgl. Kapitel 34.6 in [Alde93]). Derartige Konzepte können helfen, sowohl bei Softwaremodifikationen nicht das gesamte Produktmodell vollständig neu aufbauen zu müssen, als auch nur diejenigen Meßwerte neu zu bestimmen, die durch die Produktmodelländerung eventuell geändert werden. Besonders relevant ist diese Performance-Optimierung für die Prozesse „Überarbeitung eines Programms" und „Trendanalyse" (vgl. Kapitel 3.3).
- *Transfer des vorgestellten Meßkonzepts auf andere Softwaresysteme*: Das Problem der zunehmenden Komplexität von Softwareprodukten und die damit aufkommende Notwendigkeit der Qualitätssicherung ist nicht auf Quelltexte von Computerprogrammen beschränkt, sondern trifft ebenfalls für andere von Computer bearbeitbaren Daten zu[36]. Exemplarisch für solche Systeme soll im folgenden kurz der Transfer auf große Datenbeschreibungen in Form von *Entity-Relationship-Modellen* und auf große *Webstrukturen* angedeutet werden:
  - Für die Datenmodellierung werden – besonders bei einer geplanten Verwendung einer relationalen Datenbank – häufig große Entity-Relationship-Modelle erstellt. Mit zunehmender Größe und Einsatzdauer (vgl. Gesetz der zunehmenden Entropie, Kapitel 8) werden auch hierfür Eigenschaften relevant, die in dieser Arbeit als interne Qualität bezeichnet werden (vgl. Kapitel 2). Dies führte Anfang der neunziger Jahre für Entity-Relationship-Modelle zur einer Vielzahl sogenannter *Qualitätsframeworks* (in dieser Arbeit: Qualitätsmodelle, vgl. Kapitel 2), die verschiedene *Qualitätskriterien* (in dieser Arbeit: Qualitätsmerkmale, vgl. Kapitel 2) vorschlagen (vgl. [Mood99]). Erst in jüngster Zeit hat sich auch dort das Bewußtsein durchgesetzt, daß fixe Qualitätsmodelle nur schwierig anzuwenden sind: „*However quality criteria are not enough on their own to ensure quality in practice, because different people will generally have different interpretations of what they mean*" ([Mood99], S. 211). Als daraufhin identifizierte Einflußfaktoren (vgl. Kapitel 2.2) identifiziert Moody z.B. den *Anwender* (*Business User*), dessen Anforderungen in das Datenmodell einfließen, den *Analytiker* (*Analyst*), der für die Erstellung des Datenmodells zuständig ist, den Administrator (*Data Administrator*), der für die Konsistenz zwischen Datenmodell und anderen Datenquellen der Applikation zuständig ist, und den *Anwendungsentwickler* (*Application Developer*), der für die Umsetzung des Datenmodells auf Datenbankschemata verantwortlich ist. In dem ebenfalls von Moody vorgestellten Maßkatalog sind neben vielen, nicht automatisch bestimmbaren Maßen (z.B. Maß 12, in dem die Modelle von den Anwendungsentwicklern bzgl. der Verständlichkeit ordinal

[36] Daher wird unter Software teilweise auch sehr viel mehr als nur Computerprogramme verstanden. So definiert die IEEE z.B. Software als „[...] *Computer programs, procedures, rules, and possibly associated documentation and data pertaining to the operation of a computer system*" [IEEE90].

bewertet werden sollen, vgl. [ebd.], S. 218) auch bereits einige auf automatisch erstellbaren Produktmodellen basierende Maße vorgestellt (z.B. Maß 16: Anzahl der Entitäten und Maß 18: Anzahl von Entitäten plus Anzahl von Relationen plus Anzahl von Attributen, vgl. [ebd.], S. 220f). Nach sorgfältiger Erstellung eines angepaßten Qualitätsmodells, Definition geeigneter Produktmodelle für Entity-Relationship-Modelle und der Herleitung geeigneter Maße können die in dieser Arbeit vorgestellten Konzepte daher auch auf diesen Bereich der Software übertragen werden. Auch das generische Distanzmaß kann innerhalb der beiden Prozesse „Kopplungbasierte Kohäsionsbestimmung" (z.B. durch die durch Relationen gegebenen Kopplungen zwischen Entitäten) und „Allgemeine Kohäsionsbestimmung" (z.B. durch das Merkmal Normalformstufe, vgl. Metrik 14 in [Mood99]) sinnvoll instanziiert werden und ermöglicht damit ebenfalls Techniken wie Clusteranalyse und Visualisierung, die ihrerseits dann wieder in der Lage sind, die meßbasierte Qualitätssicherung auch von Entity-Relationship-Modellen effizient zu unterstützen.

- Ebenfalls große Webstrukturen besitzen eine zunehmend hohe Komplexität, die durch eine konsequente Qualitätssicherung besser beherrschbar wird: „*The World Wide Web is probably the best example of an information space where users need support. The structure of the web has evolved rather than been designed and the quantity of information is both very great and it is changing rapidly*" ([HDWB99], S. 504). Während viele Ansätze die Hauptprobleme in der Nutzung dieser Webstrukturen als Internetbesucher sehen und dort entsprechende, meistens ebenfalls auf Visualisierung beruhende Ansätze vorstellen (vgl. z.B. *Narcissus-System* in [HDBW99] oder das *WebBook*® in [Chen99]), ist die der in dieser Arbeit vorgestellten Konzepte eher entsprechende Sichtweise die des Webadministrators, der für die Erstellung und Pflege einer Domäne verantwortlich ist: Wie bei der Erstellung großer Softwaresysteme sind auch hier interne Qualitätseigenschaften wie z.B. Strukturiertheit (z.B. in Form der Aufteilung einzelner Seiten auf Unterverzeichnisse bzw. Domänen) oder Änderbarkeit (z.B. in Form gemeinsam genutzter Ressourcen wie etwa Hintergrundbilder) wünschenswert. Nach Etablierung eines Qualitätsmodells für Webstrukturen aus der Sicht eines Webadministrators kann wiederum ein Produktmodell erstellt werden, das die wesentlichen Daten einer Webstruktur enthält. Mögliche Entitäten sind z.B. die Domänen (z.B. <Servername>/Domäne$_1$, <Servername>/Domäne$_2$ usw.), die innerhalb einer Domäne vorkommenden HTML-Seiten mit jeweiligen Links dazwischen, die ebenfalls in einer Domäne existierenden Peripheriedaten wie z.B. Bilder oder Klänge und die Links zwischen diesen Peripheriedaten und HTML-Seiten. Neben klassischen Meßwerten wie Kopplungsmeßwerte einer Seite (z.B. Anzahl benötigter Peripheriedaten) oder Größenmeßwerte von Peripheriedaten (z.B. Anzahl Bytes) können wiederum die vorgestellten Prozesse der kopplungbasierten und allgemeinen Kohäsionsbetrachtung angewendet werden. Möglich sind u.a. Ähnlichkeiten aufgrund der Linkstruktur zwischen HTML-Seiten (ähnlich der gewichteten, methodenbasierten Interaktionskohäsion, vgl. Kapitel 8.3.2), aufgrund der Menge der verwendeten Peripheriedaten (ähnlich der gewichteten, attributbasierten Interaktionskohäsion, vgl. Kapitel 8.3.2) oder Kombinationen beider Kopplungen (ähnlich der ungewichteten Interaktionskopplung, vgl. Kapitel 8.3.2). Letztere ist in Form einer prototypischen Implementierung für eine Webstruktur in Abbildung 80 dargestellt, d.h. zwei Elemente (HTML-Seiten und Peripheriedaten) sind um so ähnlicher, je mehr gemeinsame eingehende Kopplungen (in Form von Links von HTML-Seiten) sie besitzen:

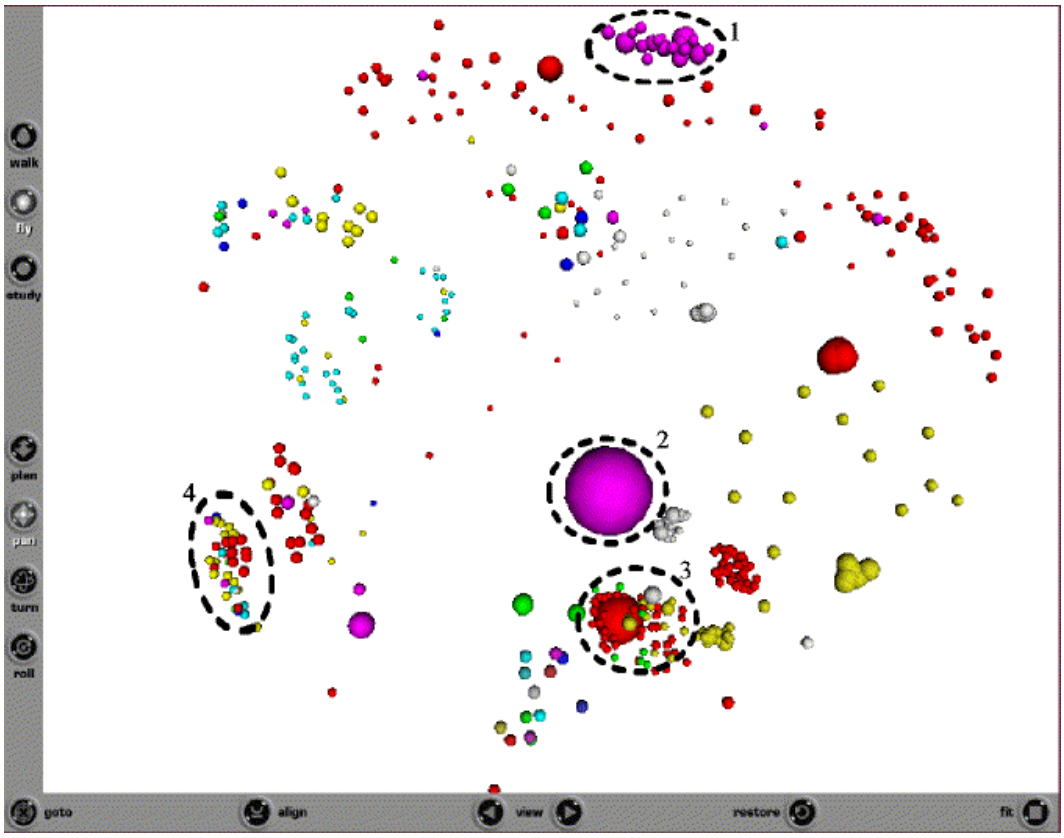


Abbildung 80: Beispiel einer Kohäsionsbetrachtung einer Webstruktur

Dargestellt sind 470 betrachtete Elemente: 230 davon sind einzelne Webseiten (HTML-Dateien) und 240 sind Peripheriedaten. Die Darstellungsparameter sind wie folgt gesetzt: Die Farbe der Elemente ist derart gesetzt, daß alle Elemente einer Domäne dieselbe Farbe besitzen. Die Größe der Elemente ist durch das Maß „Anzahl eingehender Kopplungen“ bestimmt. Innerhalb des dargestellten Screenshots sind u.a. folgende Situationen ersichtlich:

- Das größte Element (vgl. 2 in Abbildung 80) repräsentiert das Hintergrundbild, das von einem Großteil der Seiten innerhalb der Domäne und deren Unterdomänen verwendet wird und daher zentral angeordnet ist.
- Deutlich sind ebenfalls generierte Präsentationsshows erkennbar, wie sie z.B. mit Microsoft PowerPoint® generiert werden können (vgl. 1 und 3 in Abbildung 80): Die von allen Einzelseiten gemeinsam verwendeten Peripheriedaten (z.B. Graphiken für Zurück- und Vorspulen) werden als besonders große Elemente jeweils zentral dargestellt.
- Der Einstiegspunkt der Domäne (in 4) besitzt eine Vielzahl von Seiten aus unterschiedlichen Subdomänen (dargestellt durch unterschiedliche Farben), die jeweils eng miteinander gekoppelt bzw. ähnliche Peripheriedaten benötigen.

Die möglichen Prozesse für eine solche auf einem angepaßten Qualitätsmodell basierende Interpretation einer Strukturvisualisierung großer Webstrukturen, die wie die Softwarevisualisierungen vollständig automatisch erstellt werden kann, entsprechen dabei den in Kapitel 3.3 aufgeführten Prozessen für die Qualitätssicherung von Softwareprodukten.

- *Alternative Darstellungsformen der virtuellen Informationsräume*: Die neue Technik der ausschließlich durch Meßwerte bestimmten Darstellung von Softwaresystemen im

dreidimensionalen Raum ist für ihre Anwender sehr ungewohnt und läßt sich erst nach einigen Stunden Einarbeitung effizient nutzen. Die Hauptprobleme dabei sind:

- *Navigationsprobleme*: Da ein Großteil der Computererfahrung auf zweidimensionalen Computeranwendungen basiert, muß die Betrachtung und insbesondere die Navigation in dreidimensionalen Räumen i.d.R. erst erlernt werden. Dies ist allerdings aufgrund der ebenfalls primär für zweidimensionale Anwendungen konzipierten Hardware nur bis zu einem bestimmten Grad möglich: So ist sowohl die dreidimensionale Betrachtung mittels spezieller Shutter-Brillen, die aufgrund der notwendigen Frequenzhalbierung zu unruhigeren Bildern führen (vgl. Kapitel 9.1), als auch die Navigation mittels lediglich zwei Dimensionen steuernden Eingabegeräten (z.B. Maus), was in der Bedienung die häufige Umschaltung zwischen verschiedenen Navigationsmodi erfordert (vgl. Kapitel 9.1), suboptimal. Neue Techniken in der Ausgabe dreidimensionaler Daten wie z.B. 3D-Datenhelme (*Head-Mounted-Displays, HMD*) oder CAVEs (*Cave Automatic Virtual Environment*) und dreidimensionale Eingabegeräte wie z.B. 3D-Mäuse oder Datenhandschuhe (*DataGlove*) können die Navigationsprobleme deutlich reduzieren (vgl. Kapitel 10 in [Vinc95]).
- *Orientierungsprobleme*: Aufgrund der Unerfahrenheit beim Explorieren dreidimensionaler Welten und aufgrund mangelnder fixer Orientierungspunkte benötigt das Kennenlernen dargestellter Welten (inkl. der Möglichkeit der Wiedererkennung bestimmter Stellen) deutlich länger als das Kennenlernen realer Welten (z.B. eines neuen Gebäudes). Eine Möglichkeit, die Fähigkeit der effizienten Orientierung in realen Welten auf die Softwarevisualisierung zu übertragen, ist das Verwenden von *Darstellungsmetaphern*: So kann z.B. versucht werden, die Meßdaten eines Softwareprodukts als Stadt oder Landschaft darzustellen: „*The greatest advantage of using a physical world metaphor is that users can easily understand how the virtual world is structured*“ ([Chen99], S. 100). Die damit verbundenen Zwänge der Darstellung (z.B. stehen Gebäude grundsätzlich auf dem Boden, können also nicht in alle drei Dimensionen beliebig verschoben werden) würden allerdings mindestens eine Darstellungsdimension begrenzen: „*Only a small range of fluctuation is allowed in the third dimension. This is why a space is sometimes called a 2.1D or 2.5D space*“ ([Chen99], S. 107). Der durch 2½-dimensionale Darstellungen entstehende Informationsverlust der Positionen der Elemente könnte allerdings durch eine entsprechend aufgewertete Vielfältigkeit der Darstellungen (vgl. z.B. die Vielfältigkeit der *Chernoff Faces* zur Darstellung multivariater Daten, Kapitel 4.8 in [ToStSt86]) teilweise kompensiert werden.

11 Referenzen

[Alde93] Albert Alderson: „*Configuration Management*“, in John McDermid (Hrsg.): „Software Engineer’s Reference Book“, CRC Press, Seite 34/1-34/18, Cornwall, 1993

[AlKh95] E.B. Allen, T.M. Khoshgoftaar: „*Properties of Entropy and Information as Software Metrics*“, Transactions on Software Engineering, Vol. 17, No.8, Seite 751-761, August 1995

[AnCaLu98] G. Antoniol, G. Canfora, A. De Lucia: „*Estimating the Size of Changes for Evolving Object Oriented Systems: a Case Study*“, in Proceedings des 6. IEEE International Symposiums on Software-Metrics, Part 4B, 10 Seiten, IEEE-Computer-Press, 1998

[Back56] J. W. Backus: „*Program’s Reference Manual: The Fortran Automatic Coding Systems for the IBM 704 EDPM*“, IBM Corporation, New York, 1956

[Baker et al.90a] Albert L. Baker, James M. Bieman, Norman Fenton, David A. Gustafson, Austin Melton, Robin Whitty: „*A Philosophy for Software Measurement*“, in Journal of Systems and Software, Seite 277-281, Juli 1990

[Baker et al.90b] Albert L. Baker, James M. Bieman, David A. Gustafson, Austin C. Melton: „*A mathematical perspective for software measures research*“, in IEEE/BCS Software Engineering Journal, 5 (5), Seite 246-254, 1990

[Balz96] Helmut Balzert: „*Lehrbuch der Software-Technik: Software-Entwicklung*“, Spektrum Akademischer Verlag, Heidelberg, 1996

[Balz98] Helmut Balzert: „*Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung“,* Spektrum Akademischer Verlag, Heidelberg 1998

[BaNe95] Richard Bache, Martin Neil: „*Introducing metrics into industry: a perspective on GQM*“, in Norman Fenton, Robin Whitty, Yoshinori Iizuka: „Software Quality: Assurance and Measurement – A worldwide Perspective“, Seite 59-68, International Thomson Computer Press, London, 1995,

[BaRe79] Victor Basili, Robert Reiter: „*Evaluating Automatable Measures of Software Development*“, in Proceedings des Workshop on Quantitative Software Models, Seite 107-116, 1979

[BaRo88] V.R. Basili, H.D. Rombach: „*The TAME project: towards improvement-orientated software environments*“, IEEE Transactions on Software Engineering, 14(6), Seite 758-773, 1988

[Basi95] V.R. Basili: „*Applying the Goal/Question/Metric Paradigm in the Experience Factory*“, in Norman Fenton, Robin Whitty, Yoshinori Iizuka (Hrsg): „Software Quality: Assurance and Measurement – A worldwid perspective“, Seite 23-44, International Thomson Computer Press, London, 1995

[BaWe84] V.R. Basili, D. Weiss: „*A methodology for collecting valid software engineering data*“, IEEE Transactions on Software Engineering, 10(6), Seite 728-738, 1984

[Beck00] Kent Beck: „*Extreme Programming explained*“, Addison Wesley, New York, 2000

[Berk92] Karel Berka: „*Are there objective grounds for measurement procedures?*“, in C. Wade Savage, Philip Ehrlich (Hrsg.): „Philosophical and foundational issues in measurement theory“, Seite 181-194, Lawrence Erlbaum Associates Publishers, London, 1992

[Biem96] James M. Bieman: „*Metric Development for Object-Oriented Software*“, in Austin Melton (Hrsg.): „Software Measurement“, Seite 75-92, International Thompson Computer Press, 1996

[BiKa95] James M. Bieman, Byung-Kyoo Kang: "*Cohesion and Reuse in an Object-Oriented System*", in ACM Symposium on Software Reusability, Seite 259-262, 1995

[BiKa98] James M. Bieman, Byung-Kyoo Kang: "*Measuring Design-Level Cohesion*", in IEEE Transactions on Software Engineering, Vol 24, No. 2, Seite 111-124, Februar 1998

[BiOt92] James M. Bieman, Linda M. Ott: „*Effect of software changes on module cohesion*“, in Proceedings of Conference on Software Maintenance, November 1992

[BiOt93] James M. Bieman, Linda M. Ott: „*Measuring Functional Cohesion*“, Technical Report CS-93-109, Michigan Technological University, 1993

[BiOt94] James M. Bieman, Linda M. Ott: „*Measuring Functional Cohesion*“, in IEEE Transactions on Software Engineering, Vol 20, No. 8, Seite 644-657, 1994

[Boeh et al. 78] B.W. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. MacLeod, M.J. Merrit: „*Characteristics of Software Quality*“, North-Holland, Amsterdam, 1978

[Booc94] Grady Booch: „*Objektorientierte Analyse und Design*“, Addison-Wesley, Bonn, 1994

[BoDö95] Jürgen Bortz, Nicola Döring: „*Forschungsmethoden und Evaluation*“, Springer-Verlag, Heidelberg, 2. Auflage, 1995

[Bowe76] T. P. Bowen: „*Specification of software quality attributes*“, Roma Laboratory, Tech. Report RADC-TR-85-37, Vols. 1-3, New York, 1976

[BrDaWü97] Lionel C. Briand, John W. Daly, Jürgen Wüst: „*A unified framework for cohesion measurement in object-oriented systems*“, Technischer Bericht ISERN-97-05, Fraunhofer Institut für experimentelles Software-Engineering, Kaiserslautern, 1997

[BrDaWü99] Lionel C. Briand, John W. Daly, Jürgen Wüst: „A *Unified Framework for Coupling Measurement in Object-Oriented Systems*“, in IEEE Transactions on Software Engineering, Vol. 25, No. 1, Seite 91-121, Jan/Feb 1999

[BrDeMe96] L. Briand, P. Devanbu, W. Melo: „*Defining and Validating Design Coupling Measures in Object-Oriented Systems*“, Technical Report ISERN-96-08, Fraunhofer-Institut für experimentelles Software-Engineering, Kaiserslautern, 1996

[BrDeMe97] L. Briand, P. Devanbu, W. Melo: „*An Investigation into Coupling Measures for C++*“, in proceeding der 19th International Conference on Software Engineering, (ICSE)‘97, Seite 412-421, Boston, 1997

[BrMoBa98] Lionel Briand, Sandro Morasca, Victor R. Basili: „*Defining and validating measures for Object-Based High-Level Design*“, Technical Report ISERN-98-04, Fraunhofer-Institut für experimentelles Software Engineering, Kaiserslautern, 1998

[BrMoBa99] Lionel .C. Briand, Sandro Morasca, Victor R. Basili: „*An operational process for goal-driven definition of measures*“, IESE-Report 017.99/E, Fraunhofer Institut für Experimentelles Software Engineering, Kaiserslautern, März 1999

[Bung77] Mario Bunge: "*Treatise on Basic Philosophy*", Volume 3: „Ontology I – The furniture of the world", D. Reidel Publishing Company, Dordrecht-Holland, 1977

[Chen99] Chaomei Chen: „*Information visualisation and virtual environments*“, Springer-Verlag, London, 1999

[ChKe91] S. R. Chidamber, C.F. Kemerer: „*Towards a Metrics Suite for object-oriented Design*“, in Proceedings der ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) ’91, Seite 197-211, ACM, 1991

[ChKe94] S. R. Chidamber, C.F. Kemerer: „*A metrics Suite for Object Oriented Design*“, in IEEE Transactions on Software Engineering, Vol. 20, No. 6, Seite 476-493, 1994

[ChKw98] Heung Seok Chae, Yong Rae Kwon: „*A Cohesion Measure for Classes in Object-Oriented Systems*“, in Proceedings des 5th International Software Metrics Symposiums, Maryland, Seite 158-166, 1998

[ChSm91] John C. Cherniavsky, Carl H. Smith: „*On Weyuker’s Axioms for Software Complexity Measures*“, in IEEE Transactions on Software Engineering, Vol. 17, No. 6, Seite 636-638, Juni 1991

[CoGu93] Richard E. Courtney, David A. Gustafson: „*Shotgun correlations in software measures*“, in Software Engineering Journal, Seite 5-15, Januar 1993

[CoYo94] Peter Coad, Edward Yourdon: „*OOD: Objektorientiertes Design*“, Prentice Hall Verlag, München, Übersetzung des Originals von 1991, 1994

[CWHT98] Mary Campione, Kathy Walrath, Alison Huml, Tutorial Team: „*The Java™ Tutorial Continued: The Rest of the JDK™*“, Addison-Wesley Pub., New York, 1998

[DäPa98] Rolf Däßler, Hartmut Palm: „*Virtuelle Informationsräume mit VRML: Informationen recherchieren und präsentieren in 3D*“, dpunkt-Verlag, Heidelberg, 1998

[Darw91] Ian F. Darwin: „*Checking C programs with lint*“, O’Reilly, Californien, 1991

[Dask92] Michael K. Daskalantonakis: „*A practical view of software measurement and implementation experiences within Motorola*“, IEEE Transactions on Software Engineering, Vol. 18, No. 11, Seite 998-1010, November 1992

[DeDuLa99] Serge Demeyer, Stephane Ducasse, Michele Lanza: „*A hybrid reverse engineering approach combining metrics and program visualization*“, in Proceedings des Workshops „Experiences in Reengineering“ der European Conference on Object-Oriented Programming (ECOOP) 99, in Lissabon/Portugal, FZI-Karlsruhe, 1999

[Denv93] Tim Denvir: „*Discrete mathematics*“, in John A. McDermid (Hrsg.): „Software Engineer’s Reference Book“, Seite 1.1 – 1.14, CRC Press, Florida, 1993

[DFKW96] Reiner Dumke, Erik Foltin, Reinhard Koeppe, Achim Winkler: „*Softwarequalität durch Meßtools*“, Vieweg, Braunschweig, 1996

[Died99] Oliver Diedrich: „*Quellcode-Verwaltung mit dem Intlant Source Explorer*“, Kurzvorstellung in C’t 13/99, S. 48, Verlag Heinz Heise, Hannover, 1999

[DuFo97] Reiner Dumke, Erik Foltin: „*Metrics-based Evaluation of Object-Oriented Software Development Methods*“, Bericht Nr. 10/97 des Software Measurement Laboratory of Magdeburg (SMLAB), Universität Magdeburg, 1997

[DGQ86] Deutsche Gesellschaft für Qualität e.V. (DGQ): „*Software-Qualitätssicherung*“, DGQ-NTG-Schrift Nr. 12-51, VDE-Verlag, Berlin, 1986

[Drom95] R. Geoff Dromey: „*A model for Software Product Quality*“, in IEEE Transactions on Software Engineering, Vol. 21, No.2, Seite 146-162, Februar 1995

[Dude88] Duden: „*Informatik - Ein Sachlexikon für Studium und Praxis*“, Dudenverlag, Mannheim, 1988

[Dumk00] Reiner Dumke: "*Software Engineering*", zweite, erweiterte und überarbeitete Auflage, Vieweg-Verlag, Braunschweig, 2000

[DuMi92] Bohdan Durnota, Christine Mingins: "*Tree-Based Coherence Metrics in Object-Oriented Design*", in Proceedings of Technology of Object-Oriented Languages and Systems (TOOLS) Conference, Seite 489-504, 1992

[Eade84] P. Eades: „*A Heuristic for Graph Drawing*", Congressus Numerantium 41, Seite 149-160, 1984

[Eber92] Christof Ebert: „*Correspondence Visualization Techniques for Analyzing and Evaluating Software Measures*", IEEE Transactions on Software Engineering, Vol. 18, No. 11, Seite 1029-1034, November 92

[Eber95] Christof Ebert: „*Complexity traces: an instrument for software project management*", in Norman Fenton, Robin Whitty, Yoshinori Iizuka: „Software Quality: Assurance and Measurement – A worldwide Perspective", Seite 166-176, International Thomson Computer Press, London, 1995

[EdKaSc93] J. Eder, G. Kappel, M. Schrefl: „*Coupling and Cohesion in Object-Oriented Systems*", Technischer Bericht an der Universität Klagenfurt, Institut für Informatik, 1993

[EiWi99] Stephen G. Eick, Graham J. Wills: „*Navigating Large Networks with Hierachi*es", in Stuart K. Card, Jock D. Mackinlay, Ben Shneiderman: „Readings in Information Visualization: Using vision to think", Seite 207-214, Morgan Kaufmann Publishers, San Francisco, 1999

[EmDrMe98] Kahled El Emam, Jean-Normand Drouin, Walcélio Melo: „*SPICE: The theory and practise of software process improvement and capability determination*", IEEE Computer Society presss, California, 1998

[Emer84] T. J. Emerson: „*A discrimant metric for module cohesion*", in proceedings of the 7th International Conference on Software Engineering, Chicaco, Seite 294-303, 1984

[EmWo87] D.W. Embley, S.N. Woodfield: „*Cohesion and Coupling for Abstract Data Types*", in proceedings der 6th international Phoenix Conference on Computers and Communications, Arizona, 1987

[ErLe96] Karin Erni, Claus Lewerentz: „*Applying Design-Metrics to Object-Oriented Frameworks*", in proceedings des Software Metrics Symposium, Seite 64-74, IEEE Computer Society Press, 1996

[Erni96] Karin Erni: „*Anwendung multipler Metriken bei der Entwicklung objektorientierter Frameworks*", Dissertation an der Fakultät für Informatik der Universität Karlsruhe, Krehl-Verlag, Münster, 1996

[Ever93] Brian S. Everitt: „*Cluster Analysis*", 3. Auflage, Halsted Press, New York, 1993

[EvMa87] M.W. Evans, J.J. Marciniak: „*Software Quality Assurance and Management*", Wiley & Sons, 1987

[FaHa84] Ludwig Fahrmeier, Alfred Hamerle: „*Multivariate statistische Verfahren*", de Gruyter-Verlag, Berlin, 1984

[Fair85] Richard Fairley: „*Software Engineering Concepts*", McGraw-Hill, New York, 1985

[FaPoFu99] Kim M. Fairchild, Steven E. Poltrock, George W. Furnas: „*SemNet: Three-Dimensional Graphic Representations of Large Knowledge Bases*", in Stuart K. Card, Jock D. Mackinlay, Ben Shneiderman: „Readings in Information Visualization: Using vision to think", Seite 190-206, Morgan Kaufmann Publishers, San Francisco, 1999

[Fent92] Norman Fenton: „*Software Metrics: A Rigorous Approach*“, Chapman Hall, London, 1992

[Fent94] Norman Fenton: „*Software Measurement: A necessary Scientific Basis*“, in IEEE Transactions on Software Engineering, Seite 199-206, Vol. 20, No. 3, März, 1994

[FePf96] Norman E. Fenton, Share Lawrence Pfleeger: „*Software Metrics: A rigorous and practical approach*“, zweite Auflage, International Thomson Publishing, London, 1996

[FeWhIi95] Norman Fenton, Robin Whitty, Yoshinori Iizuka: „*Software Quality: Assurance and Measurement - A worldwide Perspective*“, International Thomson Computer Press, London, 1995

[FiNe01] F. Fioravanti, P. Nesi: „*A study on fault-proneness detection of Object-Oriented systems*“, in proceedings der 5th Conference on Software Maintenance and Reengineering (CSMR 2001), Lissabon/Portugal, Seite 121-130, IEEE Computer Society, 2001

[Finn et al.97] P.J. Finnigan, R.C. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H.A. Müller, J. Mylopoulos, S.G. Perelgut, M. Stanley, K. Wong: „*The software bookshelf*“, in Systems Journal, Vol 36, No. 4, IBM, 1997

[FoSc98] Martin Fowler, Kendall Scott: „*UML konzentriert: Die neue Standard-Objektmodellierungssprache anwenden*“, Addison-Wesley, Bonn, 1998

[Fowl99] Martin Fowler: „*Refactoring - Improving the design of existing code*“, Addison-Wesley, Massachusetts, 1999

[Fraz92] A. Frazer: „*Reverse Engineering - hype, hope or here ?*“ in Hall Chapman, P.A.V. Hall (Hrsg.): „Software Reuse and Reverse Engineering in Practice (Unicom Applied Information Technology, No 12)“, Seite 209-243. Chapman & Hall, New York, 1992

[FrLuSa95] Karol Frühauf, Jochen Ludewig, Helmut Sandmayr: „*Software-Prüfung - Eine Anleitung zum Test und zur Inspektion*“, 2. Auflage, Teubner-Verlag, Stuttgart, 1995

[Furn81] G. W. Furnas: „*The FISHEYE view: a new look at structured files*“, Bell Laboratories, Technical Memorandum #81-11221-9, 1981, wieder veröffentlicht in Stuart K. Card, Jock D. Mackinlay, Ben Shneiderman (Hrsg): „Readings in Informations Visualization: Using Vision to think“, Seite 312-330, Morgan Kaufmann Publishers, San Franciso, 1999

[GhJaMa91] Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli: „*Fundamentals of Software Engineering*“, Prentice-Hall International, London, 1991

[GHJV96] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: „*Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*“, Addison-Wesley, Bonn, 1996

[GiDa95] Alan E. Giles, Gregory T. Daich: „*Metrics Tools*“, in CROSSTALK: The journal of defense software engineering, Februar 95, Vol. 8, No. 2, 1995

[Gilb87] T. Gilb: „*Design by Objectives*“, North-Holland, Amsterdam, 1987

[Gill92] Alan Gillies: „*Software Quality: Theory and Management*“, Chapman & Hall Computing, London, 1992

[GiSe89] Virginia R. Gibson, James A. Senn: „*System structure and Software Maintenance Performance*“, Communications of the ACM, Vol 32, No.3, Seite 347-358, März 1989

[Glas96] Robert L. Glass: „*The Relationship between Theory and Practice in Software Engineering*“, in Communications of the ACM (CACM), Seite 11-13, Vol 39, No. 11, November 1996

[Grad92] R.B. Grady: „*Practical Software Metrics for Project Management and Process Improvement*“, Prentice Hall, Englewood Cliffs, 1992

[GrCa87] R.B. Grady, D.L. Caswell: „*Software Metrics: Establishing a company-wide program*“, Prentice Hall, Englewood Cliffs, 1987

[GrTh00] Volker Gruhn, Andreas Thiel: „*Komponentenmodelle: DCOM, JavaBeans, EnterpriseJavaBeans, Corba*“, Addison-Wesley, München, 2000

[GuTaWe93] D.A. Gustafson, J.T. Tan, P. Weaver, "*Software Measure Specificati*on", ACM Software Engineering Notes, Vol. 18, No. 5, Seite 163-168, December 1993

[HaCoNi98] Rachel Harrison, Steve J. Counsell, Reuven V. Nithi: „*An evaluation of the MOOD set of object-oriented Software metrics*“, in IEEE Transactions on Software Engineering, Vol. 24, No. 6, Seite 491-496, Juni 1998

[Hals77] Maurice H. Halstead: „*Elements of Software Science*“, Operating and Programming Systems Series, Volume 7, New York, 1977

[HDWB99] R.J. Hendley, N.S. Drew, A.M. Wood, R. Beale: „*Case Study - Narcissus: Visualising Information*“, in Stuart K. Card, Jock D. Mackinlay, Ben Shneiderman: „Readings in Information Visualization: Using vision to think“, Seite 503-512, Morgan Kaufmann Publishers, San Francisco, 1999

[HeEd93] B. Henderson-Sellers, J.M. Edwards: „*The fountain model for object-oriented systems development*“, Object Magazine, 3 (2), Seite 71-79, 1993

[HeHe95] René Henrion, Günter Henrion: „*Multivariate Datenanalyse: Methodik und Anwendung in der Chemie und verwandten Gebieten*“, Springer-Verlag Berlin, 1995

[Heir98] James T. Heires: „*Product Review - SLIM-Metrics ®: A powerful new repository and analysis tool*“, ADT-Magazine, June 1998

[HeKa81] S. Henry, D. Kafura: „*Software structure metrics based on information flow*“, IEEE Transactions on Software Engineering, Vol. 7, No. 5, Seite 510-518, 1981

[Hend96] Brian Henderson-Sellers: „*Object-oriented metrics - measures of complexity*“, Prentice Hall, New Jersey, 1996

[HeNy96] Mats Henricson, Erik Nyquist: „*Industrial strength C++: Rules and Recommendations*“, Prentice Hall, London, 1996

[Herc94] Michael Herczeg: „*Software-Ergonomie - Grundlagen der Mensch-Computer-Kommunikation*“, Addison-Wesley, Bonn, 1994

[Heus93] Harro Heuser: "*Lehrbuch der Analysis - Teil 1*", 10. Auflage, B. G. Teubner Verlag, Stuttgart, 1993

[HiMo95] M. Hitz, B. Montazeri: „*Measuring coupling and cohesion in object-oriented systems*“, in Proceedings des International Symposium on Applied Corporate Computing, Monterrey/ Mexico, Oktober 1995

[HKLR84] W. Hesse, H. Keutgen, A.L. Luft, H.D. Rombach: „*Ein Begriffssystem für die Softwaretechnik*“, in: Informatik-Spektrum, 7/1984, Seite 200-213, Springer-Verlag, Heidelberg, 1984

[Holz01] Andreas Holzinger: „*Basiswissen Multimedia - Band 2: Lernen - Kognitive Grundlagen multimedialer Informationssysteme*“, Vogel Buchverlag, Würzburg, 2001

[Hump89] W.S. Humphrey: „*Managing the Software Process*“, Addison-Wesley, MA, 1989

[IEEE90] IEEE Computer Society: „*IEEE Standard Glossary of Software Engineering Terminology*“, IEEE Std. 610.12-1990, IEEE Inc., New York, 1990

[Iizu95] Yoshinori Iizuka: „*A new paradigm for software quality: the turning point for the Japanese software industry*“, in Norman Fenton, Robin Whitty und Yoshinori Iizuka (Hrsg): „Software Quality – Assurance and Measurement, a worldwide perspective“, Seite 110-120, Thomson Computer Press, London, 1995

[ISO8402] „*Information zu DIN EN ISO 9000 - DIN EN ISO 9004, DIN EN ISO 10007, DIN EN ISO 8402, und DIN 55350-11: Normen zum Qualitätsmanagement*“, Beuth-Verlag, Berlin, 2001

[ISO9000] s. [ISO8402]

[ISO9126] „*ISO/IEC 9126: Informationstechnik; Bewertung von Software-Produkten; Qualitätsmerkmale und Richtlinien für deren Anwendung*“, Beuth-Verlag, Berlin, 1991

[JaGaRi99] Mehi Jazayeri, Harald Gall, Claudio Riva: „*Visualizing Software Release Histories: The use of color and third dimension*“ in proceedings der International Conference on Software Maintenance (ICSM), 30.8-3.9.99, Oxford, 1999

[JaLi91] Ivar Jacobson, Fredrik Lindström: „*Re-engineering of old systems to an object-oriented architecture*“, in Andreas Paepcke (Hrsg.): „Conference on Object-Oriented Programming Systems, Languages, and Applications: 6th Annual Conference, Phoenix“, S. 340-350, Sigplan Notices, Vol. 26, No. 11, November 1991

[Jone86] Capers Jones: „*Programming Productivity*“, McGraw Hill, New York, 1986

[Jone93] Capers Jones: „*Critical Problems in Software Measurement*“, IS Management Group Report, Carlsbad, Kalifornien, 1993

[Jone94] Capers Jones: „*Assessment and Control of Software Risks*“, Yourdon Press, New York, 1994

[Jone00] Capers Jones: „*Software assessment, benchmarks, and best practices*“, Addison Wesley Longman, New York, 2000

[Kahl98] Bernd Kahlbrandt: „*Objektorientierte Software-Entwicklung mit der Unified modeling language*“, Springer-Verlag, Berlin, 1998

[KaKa89] Tomihisa Kamada, Satoru Kawai: "*An algorithm for drawing general undirected graphs*", in Information Processing Letters 31, Seite 7-15, North-Holland, 1989

[Kan95] Stephen H. Kan: „*Metrics and models in software quality engineering*“, Addison-Wesley, Massachusetts, 1995

[KiPfFe95] Barbara Kitchenham, Shari Lawrence Pfleeger, Norman Fenton: „*Towards a Framework for Software Measurement Validation*“, in IEEE Transactions on Software Engineering, Vol. 21, No. 12, Seite 929-944, 1995

[KlGa95] René Klösch, Harald Gall: „*Objektorientiertes Reverse-Engineering: Von klassischer zu objektorientierter Software*“, Springer Verlag, Heidelberg, 1995

[KoMe95] Günter Koch, Richard Messnarz: „*ESPRIT 5441 BOOTSTRAP: A European assessment method for evaluating process maturity profiles and ISO requirements*“, in Monika Müllerburg, Alain Abran (Hrsg.): „Metrics in software evolution“, Seite 39-64, Oldenbourg Verlag, München 1995

[Kran et al.71] David H. Krantz, R. Duncan Luce, Patrick Suppes, Amos Tversky: „*Foundations of measurement - Volume 1: Additive and polynomial representations*“, Academic Press, New York, 1971

[Kriz88] Jürgen Kriz: „*Facts and artefacts in social science: An ephistemological and methodological analysis of empirical social science*“, Research Technique, McGraw Hill Research, 1988

[KrRo99] Jens Krinke, Torsten Robschink: „*Kombination von Slicing mit Constraint-Solving für Software-Reengineering*“, in Proceedings des Workshop Software Reengineering, Bad Honnef, Fachberichte für Informatik, 7/99, Universität Koblenz-Landau, Seite 133-140, 1999

[KuFo94]Aruna Kumar, Richard H. Fowler: „*A spring modeling algorithm to position nodes of an undirected graph in three dimensions*“, Technical Report NAG9-551-4, Department of computer science, University of Texas, 1994

[Lagu97] Bruno Laguë: „*Assessing Risks related to Software Source Code using DATRIX™*“, in Proceedings des Workshops des Quality Assurance Management Committee (QAMC, heute Committee on Communications of Quality & Reliability, CQR), 1997

[LaJaSi91] K.B. Lakshmanan, S. Jayaprakash, P.K. Sinha: „*Properties of Control-Flow Complexity Measures*“, in IEEE Transactions on Software Engineering, Vol. 17, No. 9, Seite 1289-1295, Dezember 1991

[Lanz99] M. Lanza: „*Combining metrics and graphs for object oriented reverse engineering*“, Master's thesis an der Universität Bern, 1999

[LeBe85] M.M. Lehman, L.A. Belady: „*Program Evolution*“, Academic Press, New York, 1985

[Lewe88] Claus Lewerentz: „*Interaktives Entwerfen großer Programmsysteme: Konzepte und Werkzeuge*“, Springer-Verlag, Heidelberg, 1988

[LiHe93] W. Li, S. Henry: „*Maintenance Metrics for the Object Oriented Paradigm*“, in proceedings des 1st International Software Metrics Symposium, Los Alamitos, Seite 52-60, IEEE Computer Society Press, 1993

[LiSw80] B. Lientz, E. Swanson: „*Software Maintenance Management*“, Addison-Wesley, New York, 1980

[Löff99] Silvio Löffler: „*Distanzmaßbasierte Gruppierung von Klassen*“, Diplomarbeit am Lehrstuhl Software-Systemtechnik, Technische Universität Cottbus, 1999

[LoKi94] Mark Lorenz, Jeff Kidd: „*Object-oriented metrics: a practical guide*“, Prentice-Hall Inc., New Jersey, 1994

[LRRA98] Peter Liggesmeyer, Martin Rothfelder, Michael Rettelbach, Thomas Ackermann: „*Qualitätssicherung Software-basierter technischer Systeme - Problembereiche und Lösungsansätze*“, in Informatik-Spektrum 21, Seite 249-258, Springer-Verlag, Heidelberg, 1998

[LuNe00] Andreas Ludwig, Rainer Neumann: „*Refactorings - Engineering meets Reengineering*“, in Jürgen Ebert, Bernt Kullbach, Franz Lehner (Hrsg): „2. Workshop Software Reengineering“, Fachberichte Informatik der Universität Koblenz-Landau, 8/2000, Seite 38-42, 2000

[MaCo96] Jean Mayrand, François Coallier. "*System Acquisition Based on Software Product Assessment*" Proceedings of the 18th International Conference on Software Engineering 1996, Seiten 210-219, IEEE Computer Society, 1996

[Mada99] Dinu Madau: „*Rules for Defensive C Programming*“, in Embedded Systems Programming, Vol. 12, Nr. 13, Seite 24-44, Miller Freeman Publications, San Francisco, 1999

[MaLa96] Jean Mayrand, Bruno Laguë: „*Object Oriented Architecture Assessment Using Metrics*“, in Proceedings der ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), 1996

[Marc94] John J. Marciniak (Eds): „*Encyclopedia of Software Engineering*“, John Wiley & Sons Inc., New York 1994

[Maye79] Richard P. Mayer: „*Denken und Problemlösen: Eine Einführung in menschliches Denken und Lernen*“, Springer-Verlag, Heidelberg, 1979

[MCCA76] Thomas J. McCabe: „*A Complexity measure*“, in IEEE Transactions on Software Engineering, Vol 2, No.4, Seite 308-320, Dezember 1976

[MCCA82] Thomas J. McCabe: „*Structured Testing: A Software Testing Methodology using the Cyclomatic Complexity Metric*“, NBS Special Publication, Washington, 1982

[McRiWa77] J.A. McCall, P.K. Richards, G.F. Walters: „*Factors in Software Quality*“, Rome Air development Center, Rom, 1977

[Meye97] Bertrand Meyer: „*Object-oriented software construction*“, 2nd edition, Prentice Hall PTR, New Jersey, 1997

[Mood99] Daniel L. Moody: „*Metrics for Evaluating the quality of entity relationship models*“, in proceedings der 17th International Conference on Conceptual Modeling in Singapore, Seite 211-225, LNCS Springer Verlag, 1999

[Mosi96] W. Richard Mosig: „*Software Review - Micro Planner X-Pert for Windows*“, in Cost Engineering, Vol 38, No.5, AACE-Publications (association for the advancement of cost engineering), 1996

[Much92] Hans-Joachim Much: „*Clusteranalyse mit Mikrocomputern*“, Akademie Verlag GmbH, Berlin, 1992

[MüKl88] H. Müller, K. Klashinsky: „*Rigi - A system for programming-in-the-large*“, in Proceedings of the 10th International Conference on Software Engineering (ICSE), Seite 80-86, IEEE Computer Society Press, April 1988

[Müll97] Bernd Müller: „*Reengineering - eine Einführung*“, Teubner-Verlag, Stuttgart, 1997

[Munz98] Tamara Munzner: „*Exploring large graphs in 3D Hyperbolic Space*“, IEEE Computer Graphics and Applications, Volume 18, No. 4, Seite 18-23, 1998

[Myer76] Glenford J. Myers: „*Software Reliability: Principles and Practices*“, John Wiley Publications, New York, 1976

[OBKM95] Linda M. Ott, James M. Bieman, Byung-Kyoo Kang, Bindu Mehra: „*Developing Measures of Class Cohesion for Object-Oriented Software*“, in proceedings des 7th Annual Workshops on Software Metrics, Juni 1995

[OmHa92] Paul Oman, Jack Hagemeister: „*Construction and Validation of Polynomials for Predicting Software Maintainability*“, Report #92-06TR, Software Engineering Test Lab, University of Idaho, 1992

[OmPf97] Paul Oman, Shari Lawrence Pfleeger (Hrsg.): “*Applying software metrics*”, IEEE Computer Society Press, Los Alamitos/California, 1997

[OmWe95] Paul W. Oman, Kurt D. Welker: „*Software Maintainability Metrics Models in Practise*“, in CrossTalk: The Journal of Defense Software Engineering, veröffentlicht vom Software Technology Support Center, Vol.8, No.11, Nov/Dez 1995

[OtTh89] Linda M. Ott, Jeffrey J. Thuss: „*The relationship between slices and Module Cohesion*“, in proceedings der 11th International Conference on Software Engineering, Pittsburgh, May 15-18, 1989

[OtTh93] Linda M. Ott, Jeffrey J. Thuss: „*Slice Based Metrics for Estimating Cohesion*“, in proceedings des IEEE CS International Software Metrics Symposiums, Baltimore, 1993

[PaSi94] Bernd-Uwe Pagel, Hans-Werner Six: „*Software Engineering: Band 1: Die Phasen der Softwareentwicklung*“, Addison-Wesley Publishing, Bonn, 1994

[Perr87] William E. Perry: „*Effective Methods of EDP Quality Assurance*“, 2. Auflage, Prentice-Hall, Englewood Cliffs, 1987

[Petr98] Roland Petrasch: „*Einführung in das Software-Qualitätsmanagement: Software-Qualität, Software-Qualitätsmanagement, Normen und Standards, DIN ISO 9000ff., V-Modell, Umsetzungsbeispiele für Verfahrensanweisungen*“, Logos Verlag, Berlin, 1998

[Pfan71] Johann Pfanzagl: „*Theory of Measurement*“, Physica-Verlag, Würzburg, 1971

[Pfei97] Andreas Pfeiffer: „*SNiFF+: eine einheitliche Arbeitsumgebung für große Softwareprojekte*“, in OBJEKTspektrum März/April 97, SIGS Conferences, Bergisch-Gladbach, Seite 30-34, 1997

[PfMc90] S.L. Pfleeger, C.L. McGowan: „*Software metrics in a process maturity framework*“, Journal of Systems and Software, Vol. 12, No. 3, Seite 255-261, 1990

[Pigo96] Thomas M. Pigoski: „*Practical Software Maintenance: Best practices for managing your software investment*“, John Wiley & Sons, New York, 1996

[PKSH95] Kevin Pulford, Annie Kuntzmann-Combelles, Stephen Shirlaw, Katherine Harutunian (Hrsg) „*A Quantitative Approach to Software Management : The Ami Handbook*“, Addison-Wesley, New York, 1995

[PoDe00] Geert Poels, Guido Dedene: „*Distance-Based Software Measurement: Necessary and Sufficient Properties for Software Measures*“, in Information and Software Technology, Vol. 42, No. 1, Seite 35-46, 2000

[Pres97] Roger S. Pressman: „*Software Engineering - A practitioner's approach*“, 4th edition, McGraw-Hill Company, New York, 1997

[Przy99] Michael Przybilski: „*Integration of metrics visualization into a software development environment*“, Diplomarbeit am Lehrstuhl Software-Systemtechnik der Technischen Universität Cottbus, 1999

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery: “*Numerical Recipes in C - The Art of Scientific Computing*”, second edition, Cambridge University Press, Cambridge, 1992

[PuMy97] Lawrence H. Putnam, Ware Myers: „*Industrial strength software: Effective management using measurement*“, IEEE Computer Socienty Press, Los Alamitos, 1997

[RaHaRo95] A.K. Rae, H.L. Hausen, P. Roberts (Editors): „*Software Evaluation for Certification: Principles, Practice and Legal Liability*“, McGraw Hill, International Software Quality Assurance Series, 1995

[Rich95] Gottfried Richenhagen: „*Bildschirmarbeitsplätze - Mehr Arbeitsschutz am Computer*“, Hermann Luchterhand Verlag, Berlin, 1995

[Robe79] Fred S. Roberts: „*Measurement theory with applications to decisionmaking, utility, and the Social Sciences*“, Encyclopedia of Mathematics and its applications, Addison Wesley Publishing, 1979

[RoHaSh98] Linda Rosenberg, Ted Hammer, Jack Shaw: „*Software Metrics and Reliability*“ in Proceedings des 9. International Symposium Software Reliability Engineering (ISSRE) in Paderborn, IEEE Computer Society Press, Best-Paper Award, 1998

[Rudo94] E.E. Rudolph: „*Erfahrungen beim Einsatz von Softwaremetriken in der Praxis*“, in Reiner Dumke, Horst Zuse (Hrsg.): „Theorie und Praxis der Softwaremessung“, Seite 35-44, Deutscher Universitätsverlag GmbH, Wiesbaden, 1994

[RuHa68] R.J. Rubey, R.D. Hartwick: „*Quantitative Measurement of Program Quality*“, ACM, National Computer Conference, Seite 671-677, 1968

[Rüpi94] Andreas Rüping: „*Modules in object-oriented systems*“, in Raimung Egel, Madhu Singh, Bertrand Meyer (Hrsg): „TOOLS 14 - Technology of Object-Oriented Languages and Systems“, Prentice Hall, 1994

[SaEh92] C. Wade Savage, Philip Ehrlich (Hrsg.): „*Philosophical and foundational issues in measurement theory*“, Lawrence Erlbaum Associates Publishers, London, 1992

[Scha94] H. Schaefer: „*How to assure the survival of software metric data collection*“, in Reiner Dumke, Horst Zuse (Hrsg.): „Theorie und Praxis der Softwaremessung“, Seite 45-58, Deutscher Universitätsverlag GmbH, Wiesbaden, 1994

[Schi98] Herbert Schildt: „*C++: The Complete Reference*“, 3rd edition, McGraw-Hill, Berkeley, 1998

[Schl98] Oliver Schlüter: „*VRML: Sprachmerkmale, Anwendungen, Perspektiven*“, O'Reilly-Verlag, Köln, 1998

[Schm et al.98] W.J. Schmidt, R.R. Roediger, C.S. Mestad, B. Mendelson, I.Shavit-Lottem, V. Bortnikov-Sitnitsky: „*Profile-directed restructuring of operating system code*“, in IBM Systems Journal, Vol 37, No. 2, Seite 1-26, 1998

[Schn92] Norman F. Schneidewind: „*Methodology for Validating Software Metrics*“, in Transactions on Software Engineering, Vol. 18, No. 5, Seite 410-422, 1992

[ScPo98] Heinz W. Schmidt, Margot Postema: „*Reverse Engineering and Abstraction of Legacy Systems*“, in Informatica: „An International Journal of Computers and Informatics“, Seite 359-371, Vol. 22, No. 3, 1998

[SDBP98] John Stasko, John Domingue, Marc H. Brown, Blaine A. Price (eds) :“*Software Visualisiation - Programming as a Multimedia Experience*“, MIT-Press, Massachusetts, 1998

[SiBe00] Frank Simon, Dirk Beyer: "*Considering Inheritance, Overriding, Overloading and Polymorphism for Measuring C++ Sources*", Technical Report 04/00, Computer Science Reports, Technische Universität Cottbus, Mai 2000

[SiBeLe01] Frank Simon, Dirk Beyer, Claus Lewerentz: "*Impact of Inheritance on Metrics for Size, Coupling, and Cohesion in Object Oriented Systems*", in Dumke, Abran (Eds.): „New Approaches in Software Measurement“, Lecture Notes on Computer Science 2006, Seite 1-17, Springer-Verlag, 2001

[Sieg84] A. Siegmund: “*Die richtige Einführungsstrategie - der Schlüssel zum erfolgreichen Einsatz neuer Methoden und Werkzeuge*”, in proceedings der Conference on Computer Assurance (COMPAS), 1984

[SiLe98] Frank Simon, Claus Lewerentz: „*A product metrics tool integrated into a software development environment*“, in proceedings der European Software Measurement Conference (FESMA), Seite 603-608, Belgien, 1998

[SiLö00] Frank Simon, Silvio Löffler: "*Semiautomatische, kohäsionsbasierte Subsystembildung*", in Reiner Dumke, Franz Lehner (Hrsg): "Software-Metriken: Entwicklungen, Werkzeuge und Anwendungsverfahren", Seite 153-170, Gabler Verlag, Wiesbaden, 2000

[SiLöLe99] Frank Simon, Silvio Löffler, Claus Lewerentz: "*Distance based cohesion measuring*", in proceedings der 2nd European Software Measurement Conference (FESMA) 99, Seite 69-84, Technologisch Instituut Amsterdam, 1999

[Simo96] Frank Simon: "*Summative Software-/Hardware-Evaluation anhand des Fallbeispiels 'Betriebsführungssystem im Kernkraftwerk Brunsbüttel'* ", Diplomarbeit am Lehrstuhl für Angewandte Informatik, Universität Oldenburg, 1996

[Sinc90] Murray A. Sinclair: "*Subjective Assessment*", in John R. Wilson, E. Nigel Corlett (Hrsg.): "Evaluation of human work - A practical ergonomics methodology", Taylor & Francis Ltd, Seite 58-88, London, 1990

[SiRuKö98] Frank Simon, Heinrich Rust, Gerd Köhler: "*An Assessment of large object oriented Software Systems: A metrics based process*", in proceedings des Workshops Object-Oriented Product Metrics for Software Quality Assessment der 12th European Conference on object-oriented programming (ECOOP) 98, CRIM Montreal, Seite 16-23, 1998

[SiRuKö99a] Frank Simon, Heinrich Rust, Gerd Köhler: "*Understanding object oriented software systems without source code inspection*", in proceedings des Workshops "Experiences in Reengineering" der 13th European Conference on Object-Oriented Programming (ECOOP) 99, FZI-Karlsruhe, 1999

[SiRuKö99b] Frank Simon, Heinrich Rust, Gerd Köhler: „*Softwaresichten: Eine Quellcode-Abstraktion zum Programmverstehen*", in Proceedings des Workshop „Software-Reengineering", in Bad Honnef, Fachberichte für Informatik, Universität Koblenz-Landau, 7/99, Seite 133-140, 1999

[SiRuLe00] Frank Simon, Heinrich Rust, Claus Lewerentz: „*Quality - Metrics - Numbers - Consequences*", in Reiner Dumke, Franz Lehner (Hrsg.): „Software-Metriken: Entwicklungen, Werkzeuge und Anwendungsverfahren", Seite 51-70, Gabler Verlag, Wiesbaden, 2000

[SiStLe00a] Frank Simon, Frank Steinbrückner, Claus Lewerentz: "*Multidimensionale Mess- und Strukturbasierte Softwarevisualisierung*", in proceedings des zweiten Workshop Software Reengineering, Fachberichte für Informatik 8/2000, Universität Koblenz-Landau, Seite 46-50, 2000

[SiStLe00b] Frank Simon, Frank Steinbrückner, Claus Lewerentz: „*3D-Spring-Embedder for complete graphs*", Technical Report 11/00, Computer Science Reports, Technische Universität Cottbus, 2000

[SiStLe01] Frank Simon, Frank Steinbrückner, Claus Lewerentz: „*Metrics based Refactoring*", in proceedings der 5th Conference on Software Maintenance and Reengineering (CSMR 2001), Lissabon/Portugal, Seite 30-38, IEEE Computer Society, 2001

[SoBe99] Rini van Solingen, Egon Berghout: „The Goal/question/metric Method", McGraw Hill Verlag, Berkeley, 1999

[Somm92] Ian Sommerville: „*Software Engineering*", 4th edition, Addison-Wesley Publishing Company, New York, 1992

[Stev46] S.S. Stevens: „*On the theory of Scales and Measurement*", in Science, Vol. 103, Seite 677-680, 1946

[StMyCo74] W. P. Stevens, G. J. Myers, L. L. Constantine: „*Structured Design*", in IBM Systems Journal, Vol 13, No 2, 1974

[StSn00] M. Streckenbach, G. Snelting: „*Points-To-Analysis für Java*“, in Proceedings of 2th Workshop Software Reengineering, Bad Honnef, Fachberichte für Informatik 8/2000, Universität Koblenz-Landau, Seite 65-68, 2000

[ToStSt86] S.H.C. du Toit, A.G.W. Steyn, R.H. Stumpf: „*Graphical Exploratory Data Analysis*“, Springer-Verlag, New York, 1986

[TrMa75] J.P. Tremblay, R. Manohar: „*Discrete Mathematical Structures with Applications to Computer Science*“, McGraw-Hill Book Company, New York, 1975

[ViLi00] D. Viswanathan, S. Liang: „*Java Virtual Machine Profiler Interface*“, in IBM-Systems Journal, Vol 39, No. 1, Seite 82-95, 2000

[Vinc95] John Vince: „*Virtual Reality Systems*“, Addison-Wesley, Cambridge, 1995

[Wall90] Ernest Wallmüller: „*Software-Qualitätssicherung in der Praxis*“, Carl Hanser Verlag, München, 1990

[Weic90] Reinhold P. Weicker: „*An Overview of Common Benchmarks*“, in IEEE Computer, Vol. 23, No. 12, Seite 65-75, Dezember 1990

[Weyu88] E. Weyuker: „*Evaluating software complexity measures*“, in IEEE Transactions on Software Engineering, Vol. 14, No. 9, Seite 1357-1365, September 1988

[Wöhe96] Günter Wöhe: „*Einführung in die Allgemeine Betriebswirtschaftslehre*“, 19. überarbeitete Auflage, Franz Vahlen, München, 1996

[WTMS95] Kenny Wong, Scott R. Tilley, Hausi A. Müller, Margaret-Anne D. Storey: „*Structural Redocumentation: A Case Study*“, in IEEE Software, Vol. 12, No. 1, Seite 46-54, Januar 1995

[YaKo95] Katsuyuki Yasudo, Keiko Koga: „*Product development and quality assurance in the software factory*“, in Norman Fenton, Robin Whitty und Yoshinori Iizuka (Hrsg): „Software Quality – Assurance and Measurement, a worldwide perspective“, Seite 195-205, Thomson Computer Press, London, 1995

[YoCo79] E. Yourdon, L.L. Constantine: „*Structured Design*“, Prentice Hall, Englewood Cliffs, 1979

[ZuBo85] Horst Zuse, Peter Bollmann: „*An axiomatic approach to Software Complexity Measures*“, proceedings des 3. Symposium on Empirical Foundations of information and Software Science III, Roskilde, Denmark, 1985. Neuveröffentlicht in Jens Rasmussen, Pranas Zunde (Hrsg): „Empirical Foundations of information and Software Science III“, Seite 13-20, Plenum Press, New York, 1987

[Züll98] Heinz Züllighoven: „*Das objektorientierte Konstruktionshandbuch nach dem Werkzeug- und Material-Ansatz*“, dpunkt-Verlag, Heidelberg, 1998

[Zuse98] Horst Zuse: „*A Framework of Software Measurement*“, Walter de Gruyter, Berlin, 1998

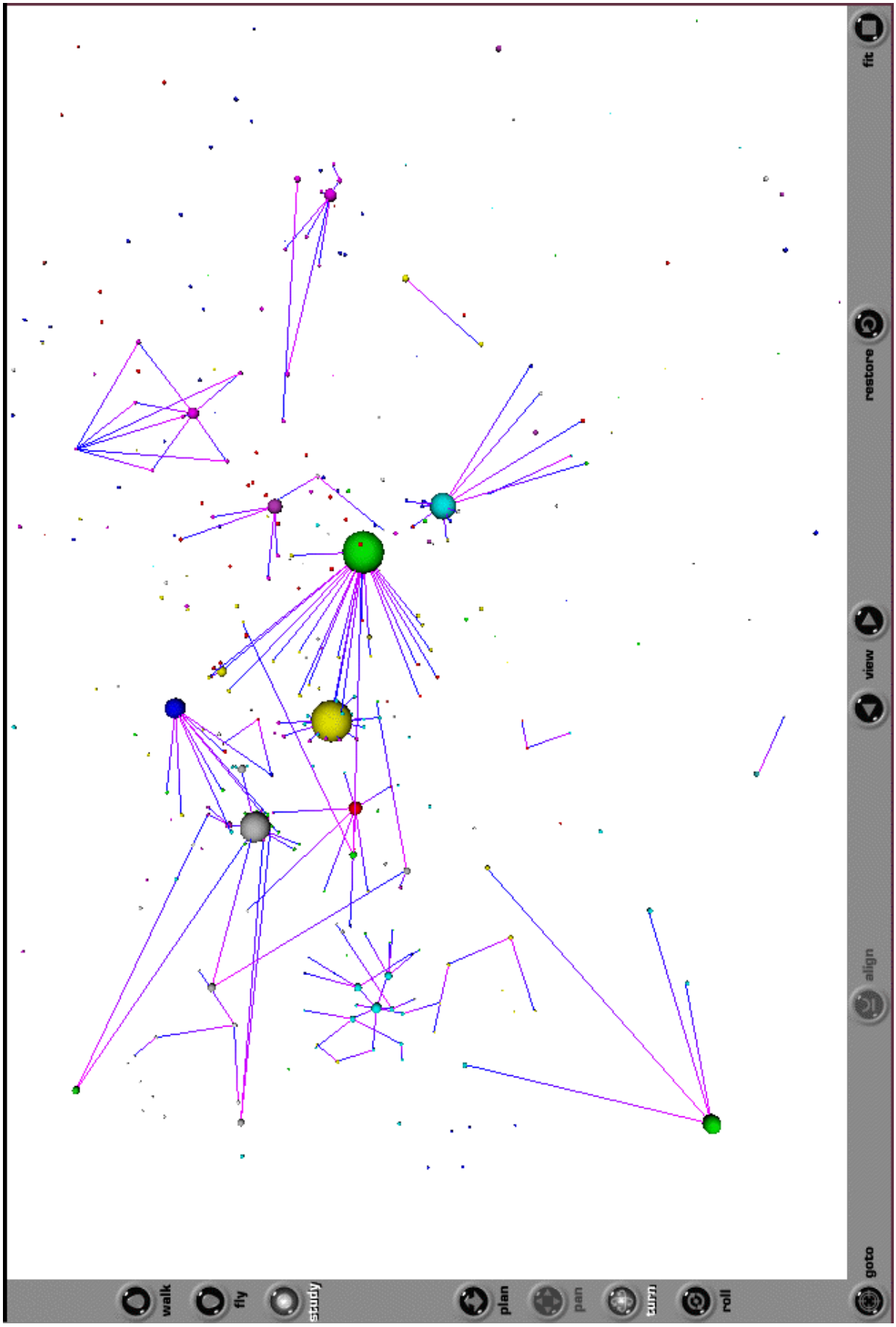
12 Anhang

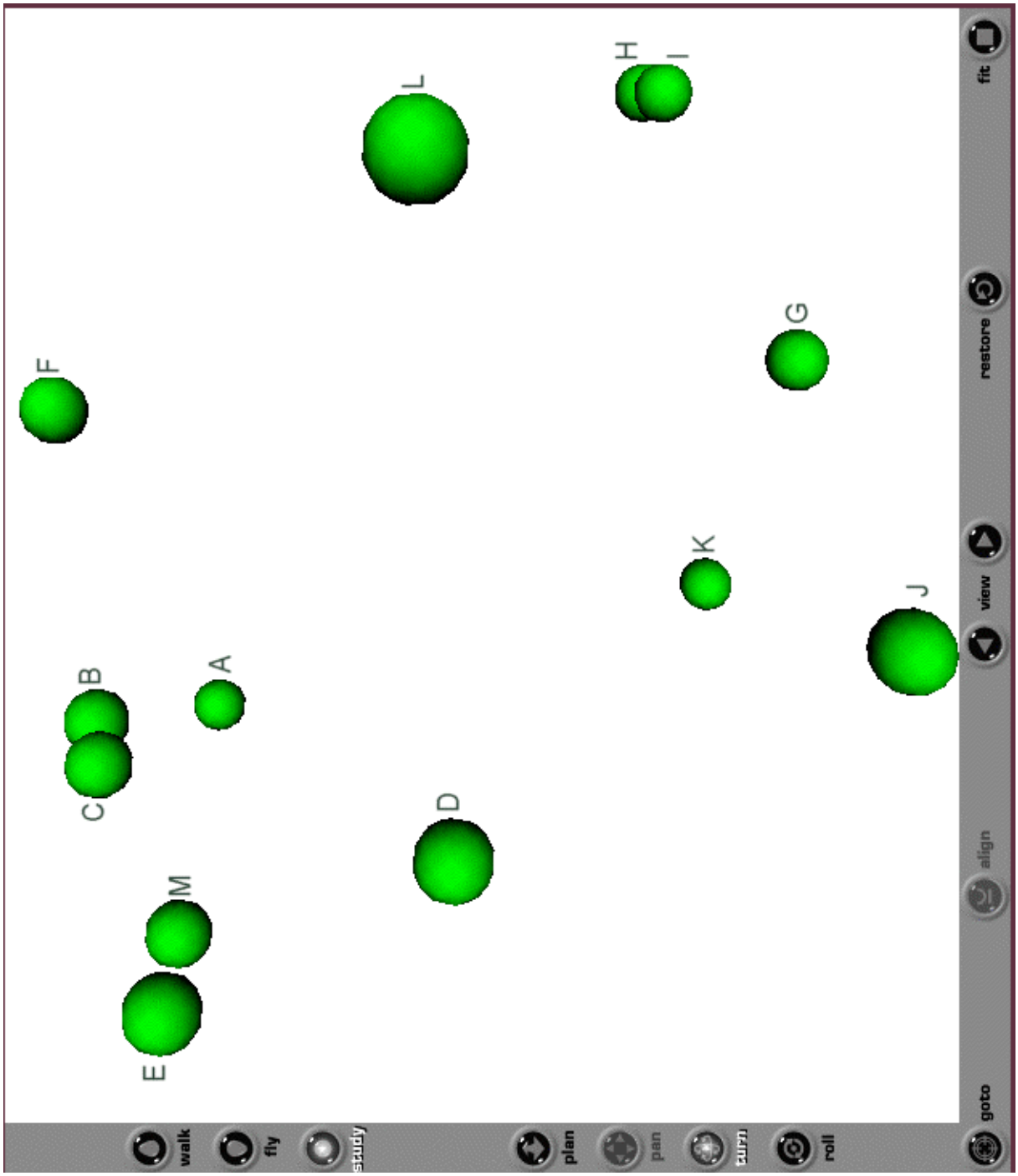
Auf den folgenden Seiten werden einige Abbildungen, die in den vorigen Kapiteln schwarz-weiß und größtenteils stark verkleinert abgebildet sind, auf Seitengröße farbig dargestellt. Die Abbildungsnummern sind dabei beibehalten worden.

Verzeichnis der ganzseitig und farbig dargestellten Abbildungen:

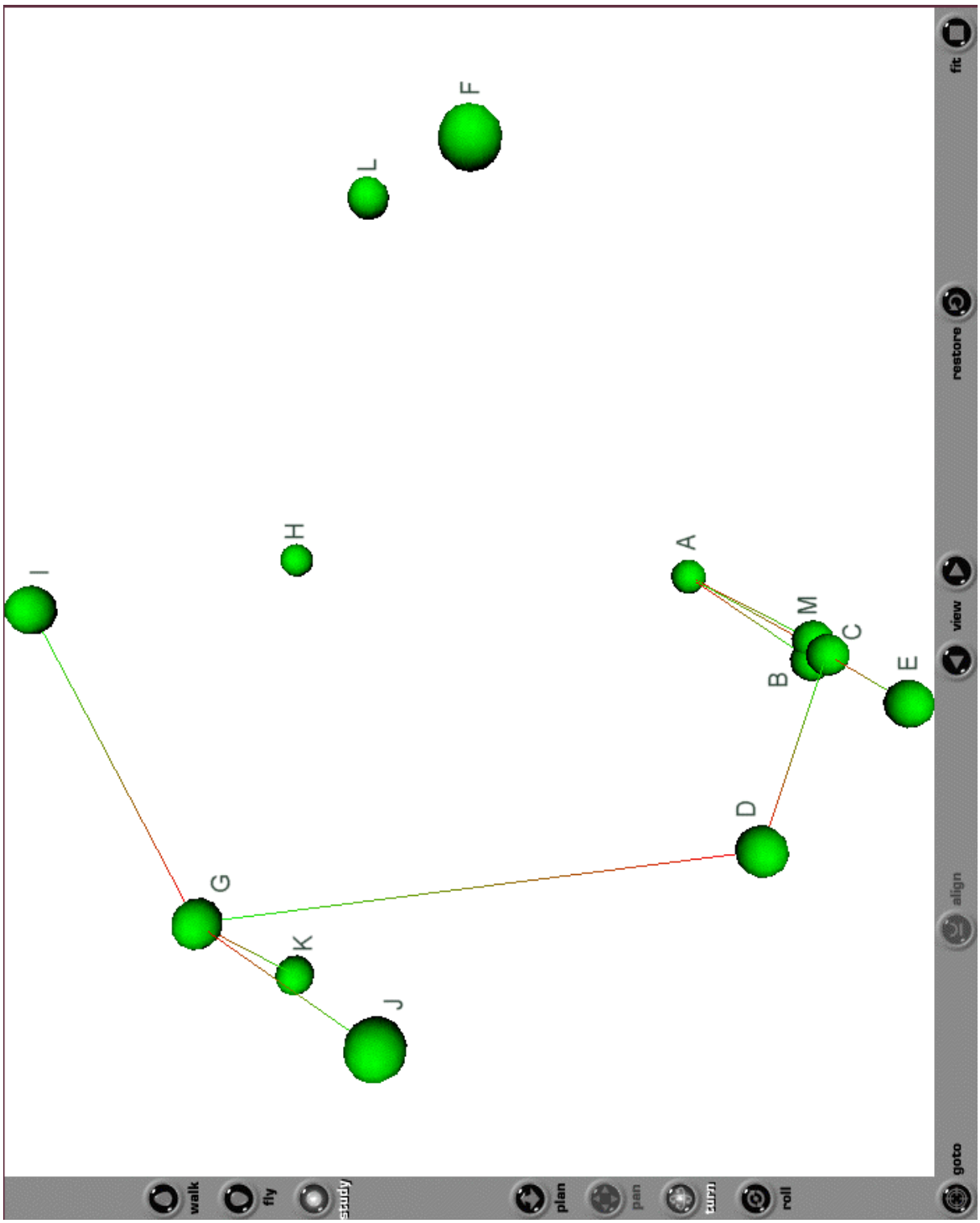
Abbildungs Nr:	Beschreibung	Seite im Text	Seite im Anhang
43	Screenshot der Betrachtung der allgemeinen Vererbungskohäsion vom Beispielsystem aus Abbildung 42.	170	272
45	Screenshot der Betrachtung der allgemeinen Vererbungskohäsion eines mittelgroßen Systems.	172	273
47	Screenshot der Betrachtung der ungewichteten Interaktionskohäsion des Systems aus Abbildung 46.	178	274
48	Screenshot der Betrachtung der gewichteten, attributbasierten Interaktionskohäsion des Systems aus Abbildung 46	180	275
49	Screenshot der Betrachtung der gewichteten, methodenbasierten Interaktionskohäsion des Systems aus Abbildung 46	182	276
50	Screenshot der Betrachtung der methoden- und attributbasierten Interaktionskohäsion des Beispielsystems.	185	277
51	Screenshot der Betrachtung der autorenbasierten Kohäsionsbetrachtung.	188	278
55	Annotierter Screenshot von CrocoCosmos	197	279
74	Beispielsituation für Refactoring „Verschieben eines Attributs“	237	280
75	Beispielsituation für Refactoring „Extrahieren einer Klasse“	238	281
76	Beispielsituation für Refactoring „Zusammenführen zweier Klassen“	239	282
77	Beispielsituation für Refactoring „Einfügen von Delegationen“	240	283
78	Beispielsituation für Refactoring „Entfernen von Delegationen“	241	284
79	Beispielscreenshot einer Klassensicht auf ein System für die Identifikation weiter zu analysierender Klassen	243	285
80	Beispiel einer Kohäsionsbetrachtung einer Webstruktur	254	286

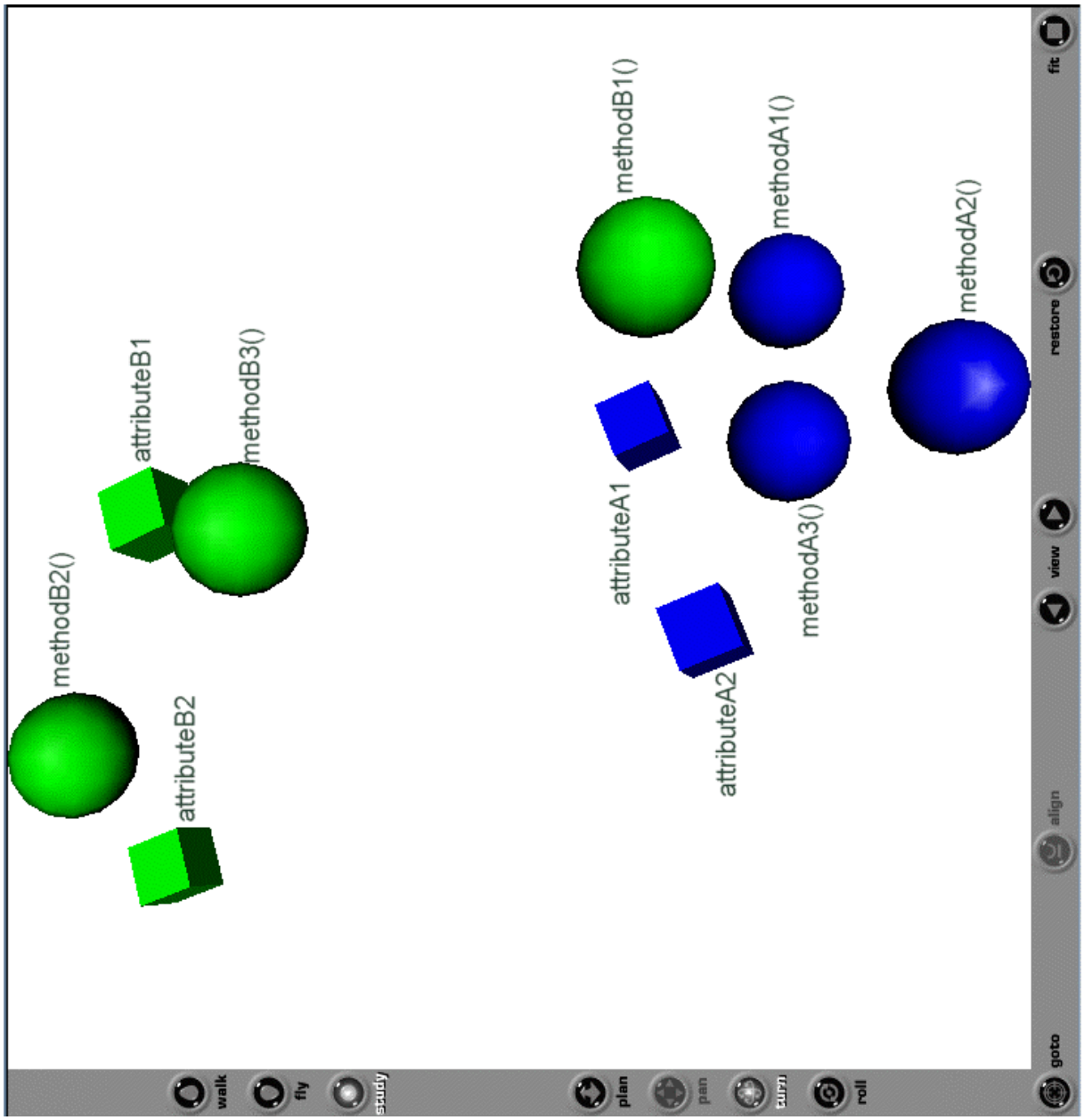
C
M I
G
A
K
F
B H
L
E
J
N
D
fit
restore
view
align
goto
walk
fly
study
plan
pan
turn
roll

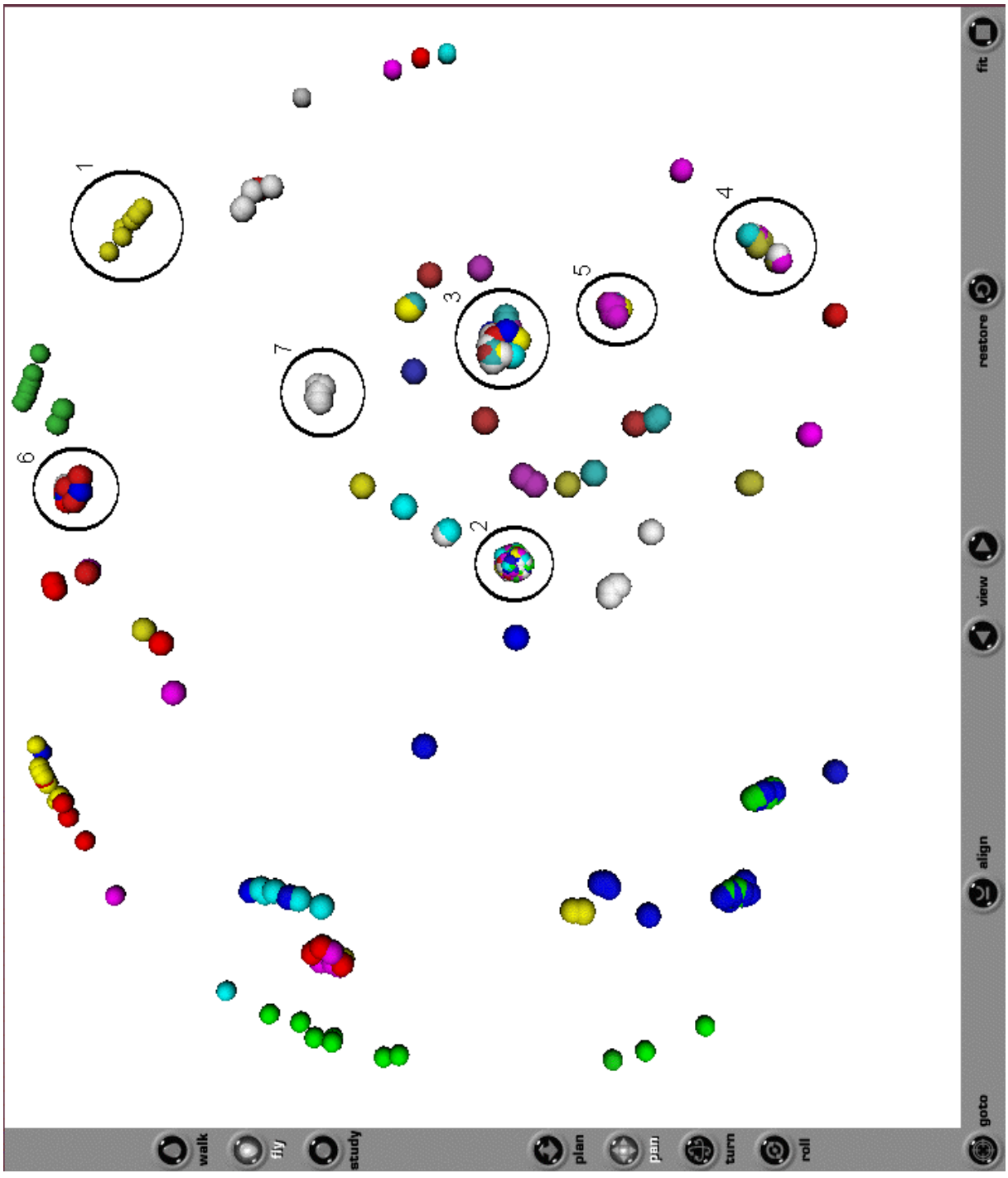

fit
restore
view
align
goto
walk
fly
study
plan
pan
turn
roll

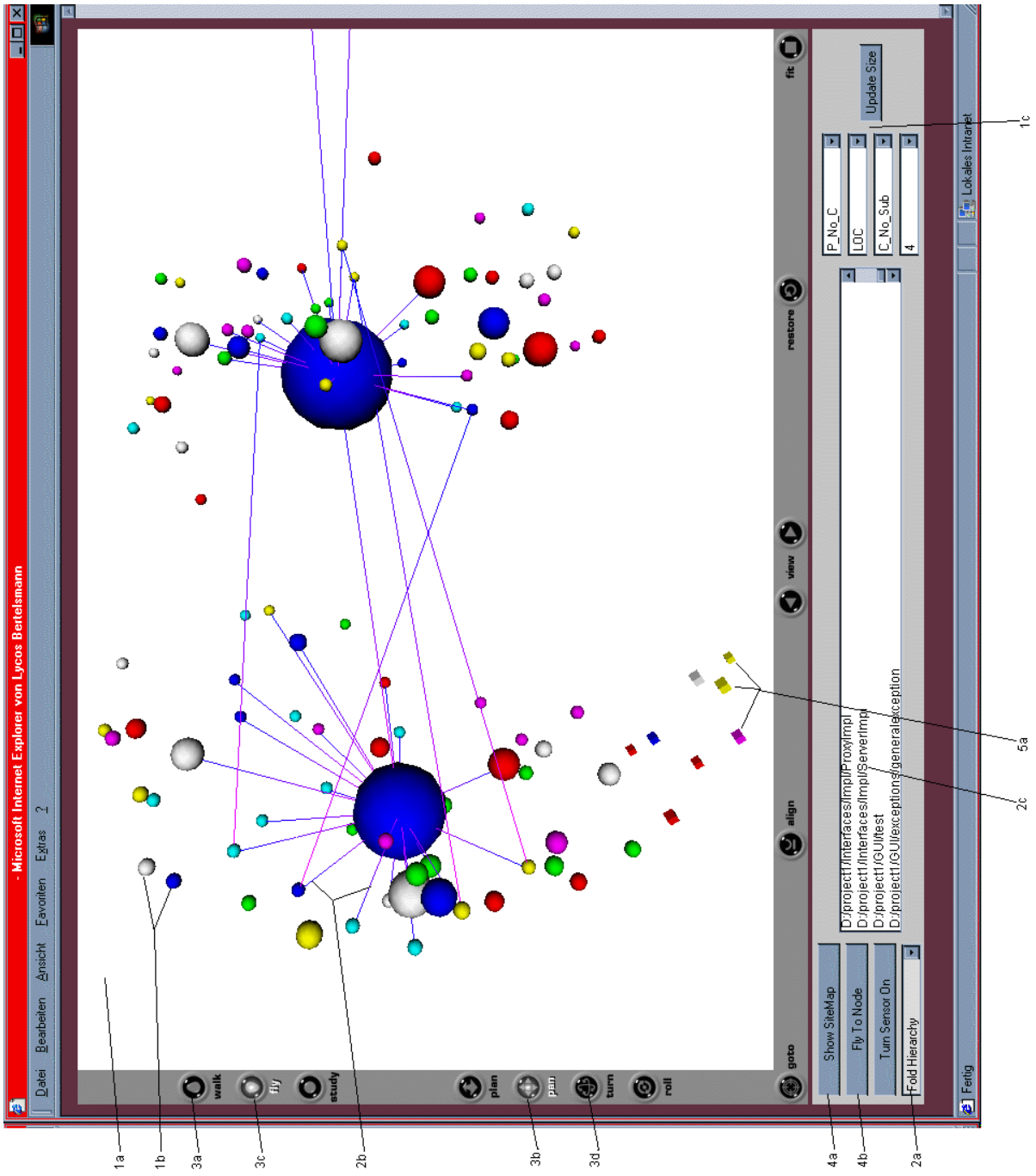

F
L
H
I
G
K
J
B
C
A
D
M
E
walk
fly
study
plan
pan
turn
roll
goto
align
view
restore
fit

K
E
D
H
G
I
J
L
A
B
C
F
M
fit
restore
view
align
goto
walk
fly
plan
pan
turn
roll


F
L
H
I
A
M
C
B
E
D
G
K
J
fit
restore
view
align
goto
walk
fly
study
plan
pan
turn
roll


methodB2()
attributeB2
attributeB1
methodB3()
attributeA1
attributeA2
methodB1()
methodA3()
methodA1()
methodA2()
walk
fly
study
plan
pan
turn
roll
goto
align
view
restore
fit


1
2
3
4
5
6
7
walk
fly
study
plan
pan
turn
roll
goto
align
view
restore
fit

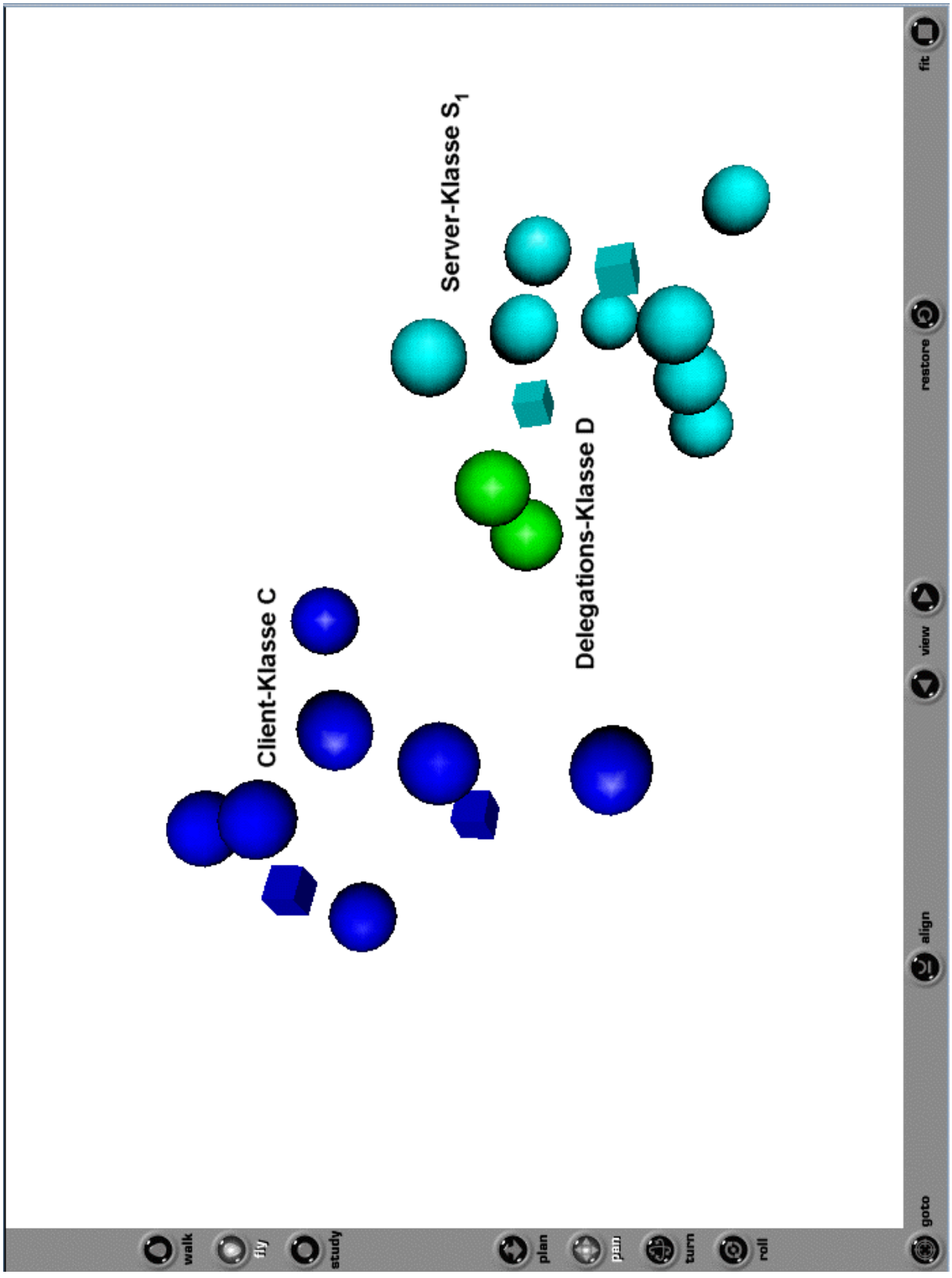

- Microsoft Internet Explorer von Lycos Bertelsmann
Datei Bearbeiten Ansicht Favoriten Extras ?
walk
fly
study
plan
pan
turn
roll
goto
align
view
restore
fit
Update Size
P_No_C
LOC
C_No_Sub
4
D:/project1/Interfaces/Impl/ProxyImpl
D:/project1/Interfaces/Impl/ServerImpl
D:/project1/GUItest
D:/project1/GUIexceptions/generalException
Show SiteMap
Fly To Node
Turn Sensor On
Fold Hierarchy
Fertig
Lokales Intranet
1a
1b
3a
3c
2b
3b
3d
4a
4b
2a
1c
5a
2c

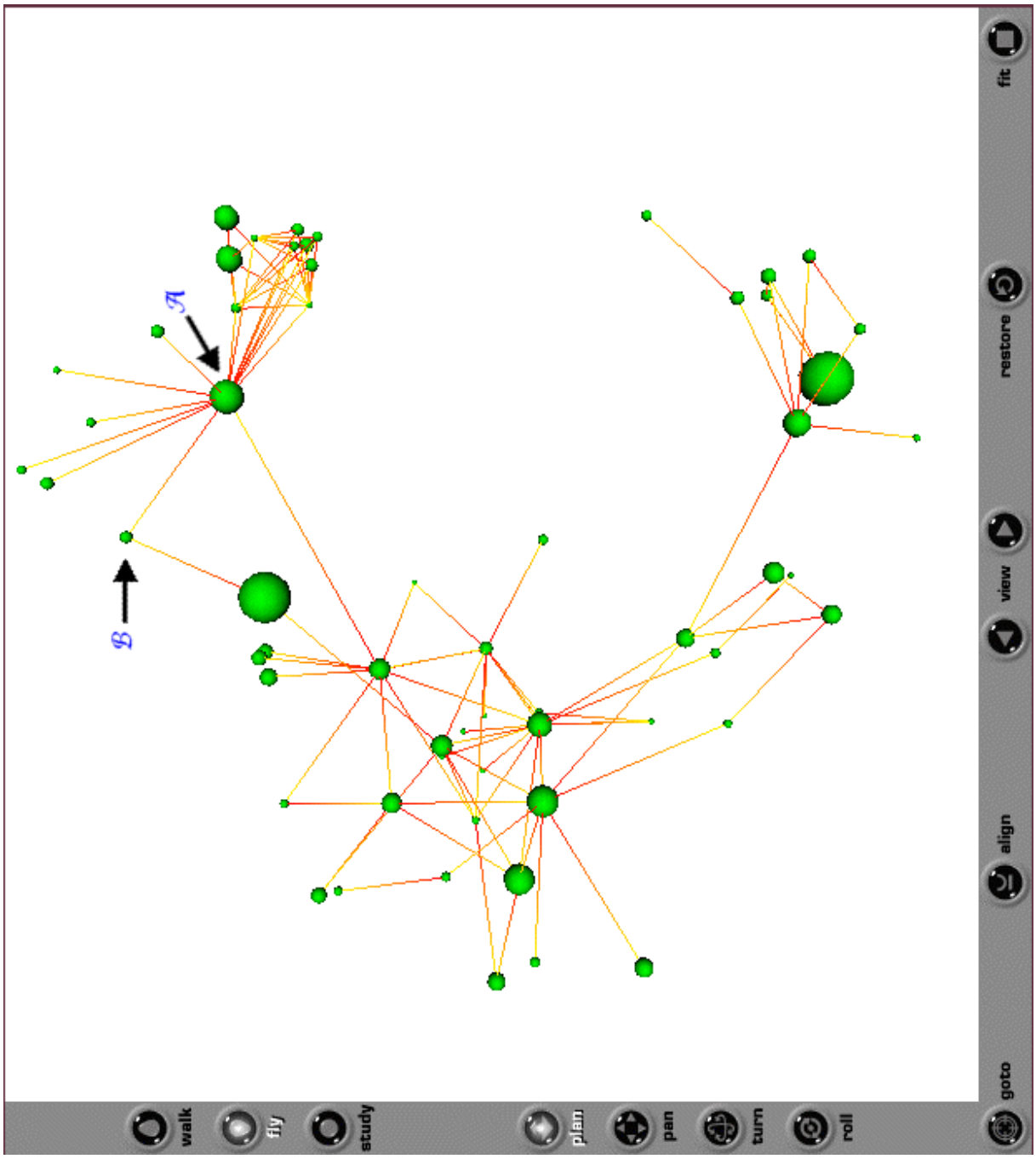
methodA1()
methodA2()
attributeA1
methodA3()
attributeB2
attributeA2
methodB1()
methodB2()
methodB3()
attributeB1
walk
fly
study
plan
pan
turn
roll
goto
align
view
restore
fit

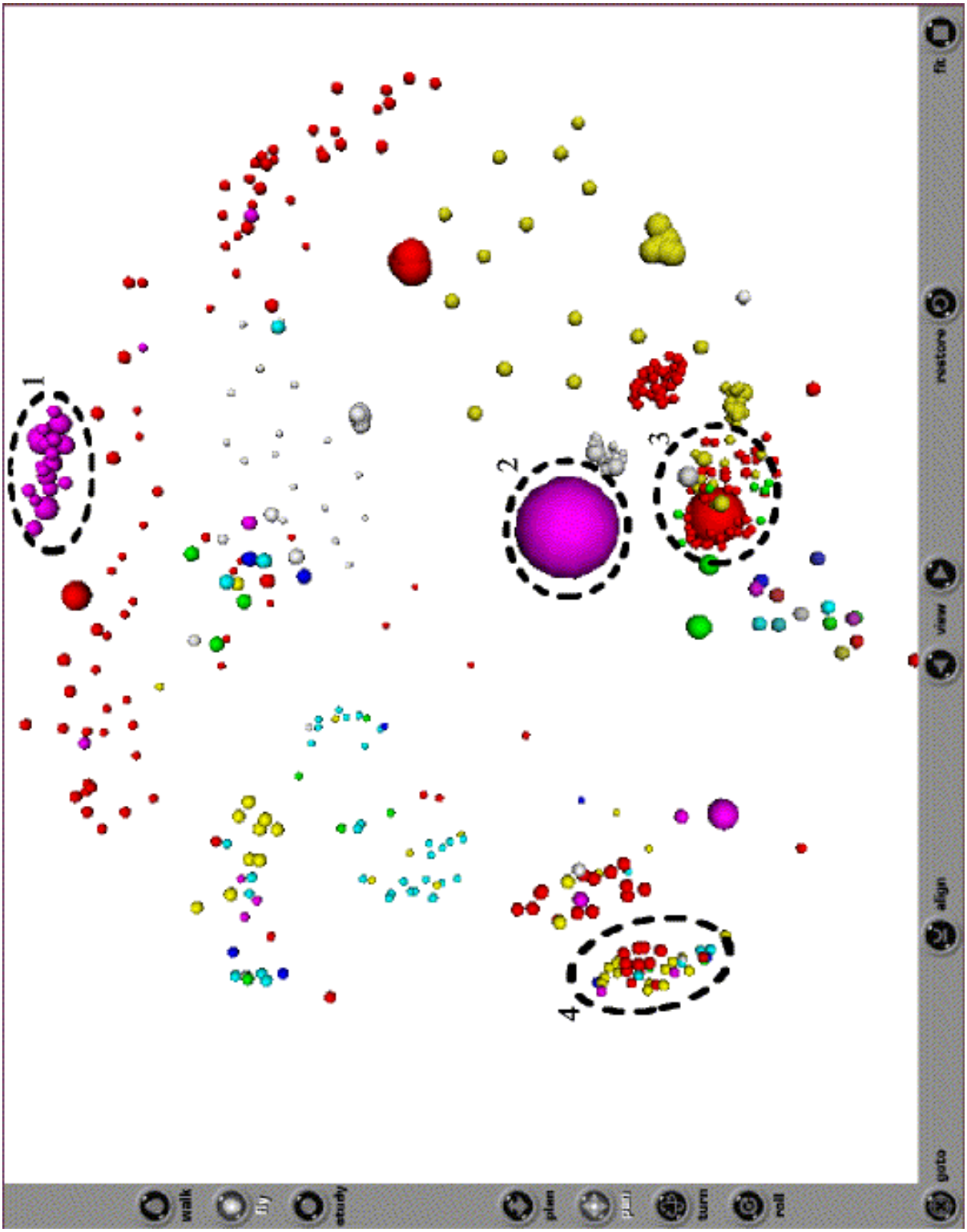
methodA9()
methodA8()
attributeA1
attributeA0
methodA7()
methodA6()
methodA5()
methodA4()
methodA3()
methodA2()
methodA1()
methodA0()
walk
fly
study
plan
pan
turn
roll
goto
align
view
restore
fit

methodA5()
attributA1
methodA2()
methodA3()
methodB2()
attributB1
methodA1()
methodB1()
attributA2
methodA6()
methodA4()
walk
fly
study
plan
pan
turn
roll
goto
align
view
restore
fit

Server-Klasse S_3
Server-Klasse S_2
Server-Klasse S_1
Client-Klasse C
walk
fly
study
plan
pan
turn
roll
goto
align
view
restore
fit


Client-Klasse C
Server-Klasse S_1
Delegations-Klasse D
walk
fly
study
plan
pan
turn
roll
goto
align
view
restore
fit


A
B
walk
fly
study
plan
pan
turn
roll
goto
align
view
restore
fit


1
2
3
4
walk
fly
study
plan
turn
roll
goto
align
view
restore
fit